Departamento de Engenharia Informática

Relatório Final de
TRABALHO FINAL DE CURSO

do Curso de
LICENCIATURA EM ENGENHARIA INFORMÁTICA E DE COMPUTADORES (LEIC)

Ano Lectivo 2004 / 2005

N.º da Proposta: 12

Título: Desenvolvimento Automático de Sistemas

Professor Orientador:

Alberto Manuel Rodrigues da Silva _____(assinatura)_____

Co-Orientador:

Miguel Luz _____(assinatura)_____

Aluno:

Nº 49673, João Paulo Pedro Mendes de Sousa Saraiva

_____(assinatura)_____

Resumo

O ProjectIT é um programa de investigação do Grupo de Sistemas de Informação (GSI) do INESC-ID, que pretende tornar a utilização de boas práticas nos projectos de TI (Tecnologias de Informação) uma tarefa automática e transparente para a equipa de desenvolvimento. Este programa visa produzir duas ferramentas, o ProjectIT-*Studio* e o ProjectIT-*Enterprise*.

Uma das vertentes do ProjectIT consiste no ProjectIT-MDD, que se baseia no conceito de desenvolvimento baseado em modelos (MDD – *Model Driven Development*). O ProjectIT-MDD vem no seguimento de dois TFCs anteriores, que tiveram como resultado a ferramenta *XIS-Tool*.

Contudo, tem surgido no contexto do ProjectIT-*Studio* um grande problema: os seus sistemas (ProjectIT-MDD, ProjectIT-RSL, etc.) conseguem comunicar, mas não conseguem "trabalhar" juntas, i.e., o nível de integração destas ferramentas está muito abaixo do que seria desejável.

Neste TFC, foi proposta uma solução: uma conversão da plataforma Eclipse (disponível em Java) para .NET. Esta plataforma oferece um mecanismo de extensibilidade poderoso, mas incentiva ao mesmo tempo a modularidade e ao encapsulamento. A plataforma foi convertida, adaptada para a ferramenta ProjectIT-*Studio*, e a ferramenta *XIS-Tool* (agora designada "ProjectIT-MDD *Tool*") foi integrada no ProjectIT-*Studio*.

Espera-se que este trabalho contribua significativamente para o esforço de desenvolvimento do ProjectIT e para a comunidade .NET. Caso a plataforma Eclipse.NET seja aceite pela comunidade .NET, pode mesmo evoluir para se tornar um *standard* no desenvolvimento de aplicações em .NET.

Palavras-chave

ProjectIT, ProjectIT-*Studio*, Eclipse, .NET, Java, integração, módulo, extensão, plataforma, ProjectIT-MDD, modelo UML, *XIS*.

Índice

LISTA DE FIGURAS VI

LISTA DE TABELAS IX

LISTA DE SIGLAS X

1 INTRODUÇÃO 1

1.1 INTRODUÇÃO 1
1.2 ENQUADRAMENTO 4
1.3 OBJECTIVOS DO TRABALHO 5
1.4 ORGANIZAÇÃO DO RELATÓRIO 7

2 ESTADO DA ARTE – CONCEITOS, TÉCNICAS E FERRAMENTAS 9

2.1 PLATAFORMA .NET 9
2.2 PLATAFORMAS PARA INTEGRAÇÃO 11
2.2.1 Plataforma Eclipse 11
2.2.2 Plataforma NetBeans 11
2.2.3 Análise e Comparação das hipóteses 12
2.3 IKVM.NET 14
2.4 *JAVA LANGUAGE CONVERSION ASSISTANT* 15
2.5 ARQUITECTURAS DE *SOFTWARE* 15
2.6 UML (*UNIFIED MODELLING LANGUAGE*) 16
2.7 LINGUAGENS XML 17
2.7.1 XML (eXtensible Markup Language) 17
2.7.2 XMI (XML Metadata Interchange) 17
2.8 MDA (*MODEL DRIVEN ARCHITECTURE*) 18
2.9 FERRAMENTAS CASE (NA PLATAFORMA .NET) 19
2.9.1 Business Change Modeller 20
2.9.2 MonoUML 20
2.9.3 Ferramentas CASE existentes em Java (ArgoUML e Thorn) 21
2.9.4 Módulo UML2 para a plataforma Eclipse 21
2.9.5 Análise e Comparação das hipóteses 21
2.10 ABORDAGEM XIS 23
2.10.1 Linguagem de Desenho XIS 25
2.10.2 Plataforma XIS 26

2.10.2.1 Geração 28
2.10.2.2 Importação 29

3 ARQUITECTURA DA PLATAFORMA ECLIPSE 30

3.1 CONCEITOS BÁSICOS 30
3.1.1 Módulo (Plugin) 30
3.1.2 Extensão (Extension) e Ponto de Extensão (Extension Point) 35
3.1.3 Plataforma (Platform Runtime) 38
3.1.4 Perfil (Profile) 38
3.1.5 Dos conceitos à plataforma 39
3.2 CONVERSÃO DA PLATAFORMA 40
3.2.1 Mecanismo de carregamento de classes 42
3.2.2 Getters/Setters vs. Propriedades 45
3.2.3 Utilização da ferramenta Log4NET 46
3.2.4 Discussão sobre a conversão entre Java e .NET 46

4 PROJECTIT-*STUDIO* 49

4.1 DESENVOLVIMENTO DO PROJECTIT-*STUDIO* SOBRE O ECLIPSE.NET 49
4.2 EVOLUÇÃO DO *XIS-TOOL* PARA O PROJECTIT-*STUDIO* 53

5 CONCLUSÕES 57

5.1 CRÍTICAS AO TRABALHO EFECTUADO 60
5.2 TRABALHO FUTURO (ECLIPSE.NET E PROJECTIT-*STUDIO*) 61
5.3 TRABALHO A DESENVOLVER NO ÂMBITO DO MESTRADO 61

6 REFERÊNCIAS 63

6.1 REFERÊNCIAS BIBLIOGRÁFICAS 63
6.2 REFERÊNCIAS NA INTERNET 64

ANEXO A MIGRAÇÃO DA PLATAFORMA ECLIPSE 68

A.1 FERRAMENTA JLCA 68
A.2 FERRAMENTA IKVM.NET 70

ANEXO B ESTADO DA CONVERSÃO DA PLATAFORMA 72

ANEXO C METODOLOGIA PROPOSTA PARA CRIAÇÃO DE MÓDULOS 77

ANEXO D ARQUITECTURAS *WINFORMS* E ECLIPSE.NET 80

D.1 ARQUITECTURA *WINFORMS* 80
D.2 ARQUITECTURA ECLIPSE.NET 81

ANEXO E SUGESTÕES PARA MELHORIAS AO ECLIPSE.NET 86

E.1 CRIAR UM FORNECEDOR DE CARREGADORES DE BIBLIOTECAS 86

E.2 ADAPTAR A PLATAFORMA À ARQUITECTURA *WINFORMS* 88

E.3 ADAPTAR A PLATAFORMA ECLIPSE.NET PARA A PLATAFORMA .NET 2.0 88

ANEXO F PADRÕES DE DESENHO NA PLATAFORMA ECLIPSE 89

F.1 *COMPOSITE* 89

F.2 *PROXY* E *BRIDGE* 92

F.3 *VISITOR* 94

F.4 *OBSERVER* 96

F.5 *STRATEGY* 98

F.6 *MEMENTO* 100

F.7 *COMMAND* 101

ANEXO G ESTRUTURA DE UM FICHEIRO DE MANIFESTO 104

Lista de Figuras

Figura 1.1 – Visão funcional do ProjectIT (extraído de [Silva2005a]). 3
Figura 1.2 – Visão aplicacional do ProjectIT (extraído de [Silva2005a]). 3
Figura 1.3 – Mapa dos conceitos relevantes deste trabalho. 8
Figura 2.1 – Componentes da plataforma .NET (extraído de [.NET]). 9
Figura 2.2 – Diagrama da arquitectura da plataforma .NET (extraído de [.NET]). 10
Figura 2.3 – Modo de funcionamento da ferramenta IKVM.NET. 14
Figura 2.4 – Arquitectura do UML (extraído de [Silva2001]). 16
Figura 2.5 – Sincronização entre modelos e código (extraído de [Lemos2003]). 19
Figura 2.6 – Sistema CASE/MDA centrado na arquitectura (extraído de [Lemos2003]). 23
Figura 2.7 – Visão geral da Abordagem XIS (extraído de [Silva2003b]). 24
Figura 2.8 – A abordagem XIS, inspirada nas ferramentas CASE e MDA (extraído de [Silva2003a]). 25
Figura 2.9 – Visão de alto nível da Plataforma XIS (extraído de [Lemos2003]). 27
Figura 2.10 – Modelo de classes para processos de geração na Plataforma XIS (extraído de [Lemos2003]). 29
Figura 3.1 – Relações de importação entre módulos. 32
Figura 3.2 – Diagrama de sequência da activação de módulos. 34
Figura 3.3 – Exemplo de um ficheiro de manifesto de um módulo (*Plugin.xml*). 35
Figura 3.4 – Relações entre Módulos e Pontos de Extensão. 36
Figura 3.5 – Declaração de um Ponto de Extensão para aplicações. 37
Figura 3.6 – Declaração de uma Extensão para o Ponto de Extensão anterior. 37
Figura 3.7 – Diagrama de alto nível dos conceitos da plataforma. 39
Figura 3.8 – Arquitectura (em vista por camadas) da plataforma Eclipse (extraído de [Eclipse]). 40
Figura 3.9 – Carregadores de Classes da plataforma Java e na plataforma Eclipse. 43
Figura 3.10 – Entidades envolvidas no mecanismo de "*AssemblyLoaders*". 45
Figura 4.1 – Esquema do ProjectIT-*Studio* e os seus módulos, sobre a plataforma Eclipse.NET. 49
Figura 4.2 – Diagrama de pacotes do ProjectIT-*Studio*. 50

Figura 4.3 – Diagrama de sequências da comunicação entre módulos. 52
Figura 4.4 – Ferramenta *XIS-Tool* com o editor de todos os processos de geração. 54
Figura 4.5 – Editor de todos os processos de geração, mas na ferramenta ProjectIT-*Studio*... 55
Figura 4.6 – Ferramenta *XIS-Tool* a mostrar alguns processos de geração. 56
Figura 4.7 – Os processos de geração da Figura 4.6, mas na ferramenta ProjectIT-*Studio*. 56
Figura 5.1 – Desenvolvimento do ProjectIT-*Studio*, antes deste trabalho. 57
Figura 5.2 – Desenvolvimento do ProjectIT-*Studio* depois deste trabalho. 59
Figura A.1 – Relatório da conversão do módulo "Core.Boot" (1/2). 68
Figura A.2 – Relatório da conversão do módulo "Core.Boot" (2/2). 69
Figura B.1 – Projectos da Plataforma no *Microsoft Visual Studio .NET*. 75
Figura D.1 – Arquitectura de uma aplicação *WinForms* na ferramenta *XIS-Tool* (extraído de [Queiroga2004]). 80
Figura D.2 – Exemplo de código para a classe principal de um Módulo. 82
Figura D.3 – Ficheiro com *strings* (*Plugin.properties*) do ficheiro de manifesto. 82
Figura D.4 – Excerto de ficheiro de manifesto (*Plugin.xml*) gerado automaticamente (1/2). . 83
Figura D.5 – Excerto de ficheiro de manifesto (*Plugin.xml*) gerado automaticamente (2/2). . 83
Figura D.6 – *Event handler*. 84
Figura D.7 – *Event listener* correspondente ao *event listener* da Figura D.6. 84
Figura E.1 – Mecanismo de fornecedor de carregadores de classes. 87
Figura F.1 – Estrutura do padrão "*Composite*" (extraído de [Gamma1995]). 89
Figura F.2 – Aplicação do padrão "*Composite*" ao conceito de "recurso" (extraído de [Gamma2003]). 90
Figura F.3 – Árvore de *widgets* de uma janela (extraído de [Gamma2003]). 91
Figura F.4 – Aplicação do padrão "*Composite*" ao SWT (extraído de [Gamma2003]). 91
Figura F.5 – Estrutura do padrão "*Proxy*" (extraído de [Gamma1995]). 92
Figura F.6 – Estrutura do padrão "*Bridge*" (extraído de [Gamma1995]). 93
Figura F.7 – Aplicação dos padrões "*Proxy*" e "*Bridge*" para acesso aos recursos (extraído de [Gamma2003]). 93
Figura F.8 – Estrutura do padrão "*Visitor*" (extraído de [Gamma1995]). 95
Figura F.9 – Aplicação do padrão "*Visitor*" aos recursos (extraído de [Gamma2003]). 95
Figura F.10 – Exemplo de implementação do padrão "*Visitor*" (extraído de [Gamma2003]). 96

Figura F.11 – Estrutura do padrão "*Observer*" (extraído de [Gamma1995]). 96
Figura F.12 – Aplicação do padrão "*Observer*" ao SWT (extraído de [Gamma2003]). 97
Figura F.13 – Estrutura do padrão "*Strategy*" (extraído de [Gamma1995]). 98
Figura F.14 – Exemplo de disposição de alguns *widgets* (extraído de [Gamma2003]). 99
Figura F.15 – Aplicação do padrão "*Strategy*" ao SWT (extraído de [Gamma2003]). 99
Figura F.16 – Estrutura do padrão "*Memento*" (extraído de [Gamma1995]). 100
Figura F.17 – Definição da interface "*IMemento*" (extraído de [Gamma2003]). 101
Figura F.18 – Estrutura do padrão "*Command*" (extraído de [Gamma1995]). 102
Figura F.19 – Partilha de uma "*Action*" por 2 *widgets* (extraído de [Gamma2003]). 102
Figura F.20 – Aplicação do padrão "*Command*" (extraído de [Gamma2003]). 103
Figura G.1 – Excerto de um ficheiro de manifesto. 104

Lista de Tabelas

Tabela 2.1 – Prós e Contras das hipóteses para ferramenta de integração 13
Tabela 2.2 – Prós e Contras das várias hipóteses para o modelador. 22
Tabela 3.1 – Sequência de activação dos módulos. 33
Tabela B.1 – Listagem dos constituintes da ferramenta Eclipse.NET. 72

Lista de Siglas

API	*Application Programming Interface*
AWT	*Abstract Window Toolkit*
CASE	*Computer Aided Systems Engineering*
CEO	Centro de Engenharia Organizacional
CLR	*Common Language Runtime*
EMF	*Eclipse Modeling Framework*
GEF	*Graphical Editing Framework*
GUI	*Graphical User Interface*
IDE	*Integrated Development Environment*
JLCA	*Java Language Conversion Assistant*
MDA	*Model Driven Architecture*
MDD	*Model Driven Development*
MOXIS	Modelo de Objectos XIS
MSIL	*Microsoft Intermediate Language*
MVC	*Model-View-Controller*
OMG	*Object Management Group*
OSGi	*Open Services Gateway initiative*
SWT	*Standard Widget Toolkit*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
RSL	*Requirements Specification Language*
UML	*Unified Modelling Language*
URL	*Uniform Resource Locator*

W3C	*World Wide Web Consortium*
XIS	*eXtensible Interactive Systems*
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*
XP	*eXtreme Programming*
XSD	*XML Schema Definition*

1 Introdução

1.1 Introdução

A actividade de concepção, implementação, gestão, operação e manutenção de sistemas de informação apresenta semelhanças evidentes com outras engenharias, tais como engenharia civil, química, ou mecânica. Todos estes projectos envolvem planeamento e gestão de recursos financeiros, gestão de pessoas e equipamentos, gestão de prazos, de qualidade, do âmbito, etc.

Existem, contudo, algumas diferenças específicas e exclusivas da área das tecnologias de informação, que importa expressar:

- A **alteração de requisitos** por parte dos clientes e/ou utilizadores **decorre de forma bastante mais frequente e exigente** face às outras engenharias. Por exemplo, não é frequente, em projectos de engenharia civil, que após a construção de uma ponte, os cidadãos (i.e., os utilizadores) e a autarquia (i.e., o cliente) exija a sua alteração!
- **É mantido ao longo de todo o projecto o mesmo tipo de representação de artefactos**: uma representação digital. Tal não ocorre manifestamente nas outras engenharias. Por exemplo, na engenharia civil, uma ponte é projectada e concebida através de múltiplas facetas segundo uma representação digital (i.e., vários desenhos e cálculos mantido num sistema de computador), mas na fase de construção propriamente dita a ponte é implementada à custa de tijolo, cimento, ferro, etc. Na generalidade das engenharias existe um corte efectivo e evidente de representação dos artefactos entre a fase de concepção e a de implementação.

Facilmente se constata que enquanto a primeira diferença é uma desvantagem e desafio significativo, já a segunda apresenta uma tremenda vantagem dos projectos de tecnologias de informação (TI) face aos de outras engenharias.

Neste âmbito, surgiu, no contexto do Grupo de Sistemas de Informação do INESC-ID, o programa de investigação "ProjectIT" [INESC-ID], cujo objectivo é a análise, integração e suporte das melhores práticas da gestão e implementação de projectos, tendo em consideração as suas especificidades particulares.

O programa "ProjectIT" (doravante referido apenas como ProjectIT) apresenta uma razoável massa crítica, atendendo ao facto de integrar o conhecimento adquirido pelo grupo em vários projectos ao longo dos últimos anos, de entre os quais se destaca, no âmbito deste relatório, a vertente relacionada com ambientes para desenvolvimento de aplicações interactivas baseado em modelos, i.e., XIS (*eXtensible Interactive Systems*) [Silva2003a] [Silva2003b]. O ProjectIT pretende produzir vários resultados, nomeadamente:

1. uma ferramenta colaborativa, com interface *Web*, designada por "ProjectIT-*Enterprise*";
2. uma ferramenta, com interface *Windows*, para realização de actividades de maior produtividade, designada genericamente por "ProjectIT-*Studio*".

A versão *Studio* tem como principal objectivo providenciar mecanismos de elevada **produtividade** ligados à gestão e especificação de requisitos, desenho de modelos, geração automática de código e desenvolvimento de software. Por outro lado, a versão *Enterprise* providencia mecanismos de **colaboração** para equipas de desenvolvimento de média e grande dimensão, privilegiando actividades ligadas à gestão do projecto e à gestão documental, tais como controlo de versões, gestão do tempo, da qualidade, ou do risco [Silva2004].

Ortogonalmente à questão das ferramentas (i.e., independentemente de ser a ferramenta ProjectIT-*Enterprise* ou ProjectIT-*Studio*), o ProjectIT integra, de forma modular e segundo uma perspectiva funcional, diferentes sistemas, destacando-se:

- ProjectIT-*Workbench*;
- ProjectIT-*Time*;
- ProjectIT-*Requirements*;
- ProjectIT-*Tests*;
- ProjectIT-MDD *(XIS)*.

A Figura 1.1 mostra os sistemas referidos e as suas dependências, segundo uma visão funcional do ProjectIT. A Figura 1.2 mostra uma visão aplicacional do ProjectIT, focando não as diferentes funcionalidades do ProjectIT mas as suas ferramentas (ProjectIT-*Studio* e ProjectIT-*Enterprise*).

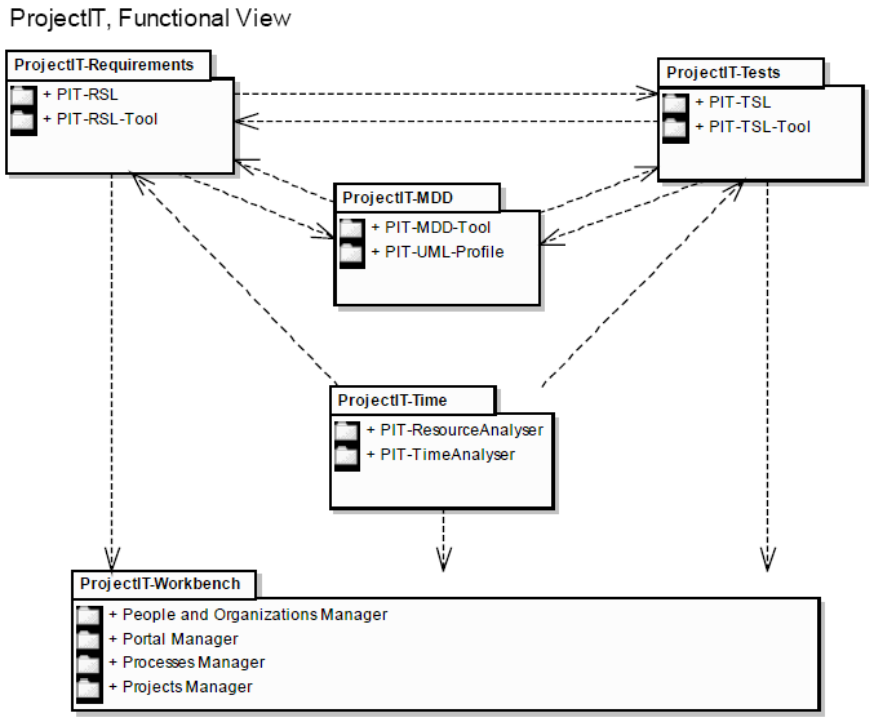

Figura 1.1 – Visão funcional do ProjectIT (extraído de [Silva2005a]).

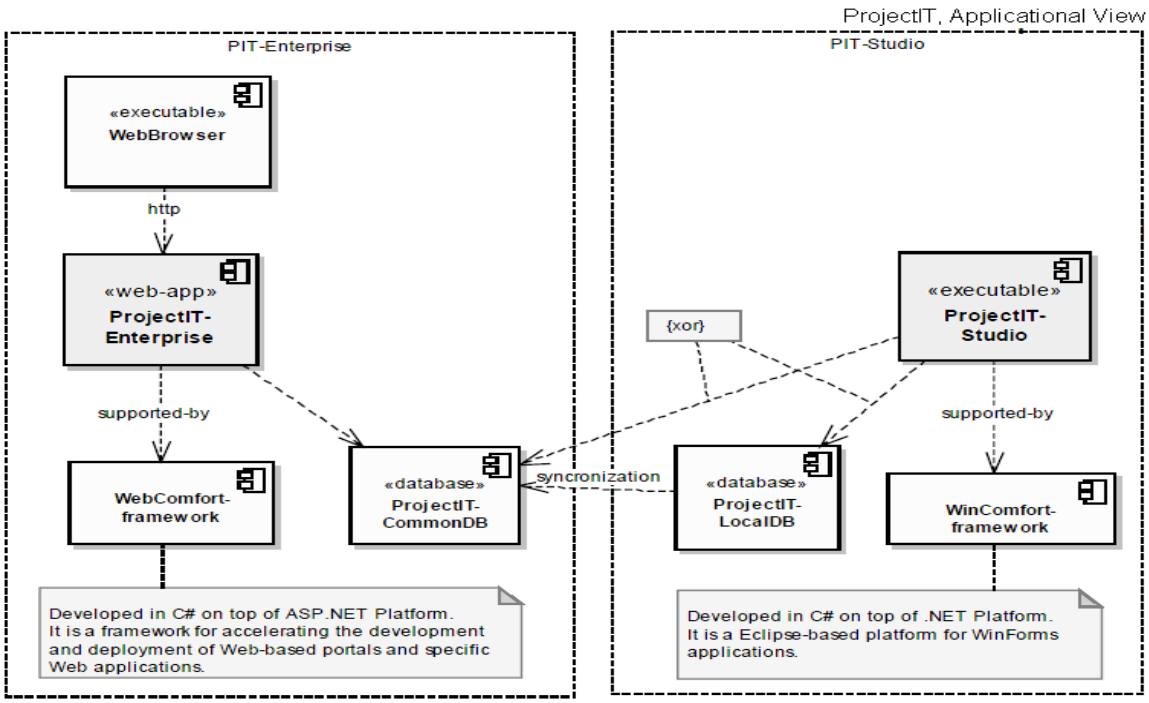

Figura 1.2 – Visão aplicacional do ProjectIT (extraído de [Silva2005a]).

Como exemplo de um dos sistemas do ProjectIT-*Studio* temos o ProjectIT-MDD, que se enquadra em termos gerais na área de modelação de sistemas de informação e na área de desenvolvimento de software baseado em modelos (MDD – *Model Driven Development*) [MDA], tendo como nome alternativo a designação "XIS", acrónimo de "*eXtensible Interactive Systems*" [Silva2004].

1.2 Enquadramento

Contudo, começa a emergir no âmbito do ProjectIT-*Studio* um grande problema: os seus sistemas podem comunicar uns com os outros (por exemplo, através de ficheiros com os dados produzidos num sistema e a serem carregados noutro sistema), mas não trabalham juntos, porque a equipa de desenvolvimento dos sistemas do ProjectIT-*Studio* não tem uma base de desenvolvimento comum em que se possa apoiar.

É no contexto deste problema que surge a necessidade de uma plataforma que permita um maior nível de integração das suas funcionalidades, que promova a modularidade desses mesmos sistemas, e que lhes permita trabalhar sobre o perfil ProjectIT/UML [Silva2005a] (também designado como perfil XIS/UML [Silva2004]). Este consiste num perfil UML (*Unified Modelling Language*) [UML] [Silva2005b] definido de acordo com os princípios e o contexto do *ProjectIT*, com vista a permitir especificar, simular e desenvolver sistemas de informação segundo a abordagem MDD. O perfil ProjectIT/UML é a linguagem comum que permite estabelecer pontes entre os vários sistemas do ProjectIT [Silva2004].

Como exemplo da integração pretendida, temos o caso de uma ferramenta CASE (*Computer Aided Systems Engineering*) [Silva2001], que suportasse o perfil ProjectIT/UML e que estivesse integrada com um gerador automático de código, a ferramenta *XIS-Tool* (desenvolvida no âmbito de um anterior TFC [Queiroga2004]). Esta ferramenta permitiria ao utilizador fazer o modelo de um sistema e, posteriormente, gerar o código fonte correspondente ao modelo especificado. O modelador e o gerador de código deverão estar integrados sob a mesma plataforma, de modo a que o processo de geração de código a partir do modelo seja o mais fluído possível (i.e., o utilizador possa executar todo este processo sem ter que iniciar uma aplicação para a ferramenta CASE e outra aplicação para o gerador de código). Para além da fluidez do processo de geração de código, uma integração a este nível

permitiria a passagem do modelo directamente entre ferramentas, o que evita todo o trabalho intermédio (propenso a erros) de guardar e carregar o modelo em ficheiros.

Outro exemplo seria a utilização simultânea de vários sistemas em conjunto. Suponha-se a existência, no contexto do ProjectIT-*Studio*, de um "*wizard*" para a especificação de um sistema computacional; este apresentaria primeiro ao utilizador um conjunto de ecrãs relativos à especificação dos requisitos (com o ProjectIT-RSL – "*Requirements Specification Language*" [Silva2004]), seguido de outro conjunto de ecrãs relativos à criação do modelo do sistema (com o ProjectIT-MDD), seguido de outros ecrãs relativos a outros sistemas do ProjectIT-*Studio*. Se o ProjectIT-*Studio* consistisse num conjunto de ferramentas isoladas (i.e., um conjunto de aplicações que funcionassem separadamente umas das outras), este cenário não seria possível, pois o "*wizard*" teria que estar, de alguma forma, presente em todos os sistemas. Caso este "*wizard*" fosse uma aplicação, teria que conter algumas características de todos os sistemas instalados no ProjectIT-*Studio*, o que iria quebrar o encapsulamento e complicar a instalação e/ou remoção de sistemas. Para além disso, caso houvessem múltiplas aplicações deste género, cada uma com requisitos semelhantes ao do "*wizard*" apresentado, a criação e manutenção do ProjectIT-*Studio* (i.e., a criação e manutenção das aplicações e bibliotecas auxiliares, algumas alterações num sistema que teriam que ser propagadas a todas as aplicações, etc.) tornar-se-ia uma tarefa caótica.

1.3 Objectivos do Trabalho

Pretendia-se inicialmente neste trabalho o desenvolvimento de uma ferramenta de modelação gráfica, para o perfil ProjectIT/UML, integrada com a ferramenta de geração de código *XIS-Tool* (aconselha-se, para uma descrição mais aprofundada do assunto, a consulta dos relatórios de TFC "*The XIS CASE Tool*" [Queiroga2004] e "*Projecto XIS – Abordagem e Ferramenta de Desenvolvimento (Case Tool e UML)*" [Lemos2003]). Entretanto, alargou-se o âmbito deste trabalho ao problema que entretanto emergia: a deficiente integração entre as ferramentas desenvolvidas nos contextos dos vários sistemas do ProjectIT-*Studio*.

Assim, o objectivo principal deste trabalho passou a ser a concepção e desenvolvimento de uma plataforma sobre a qual se possam desenvolver os vários sistemas constituintes do ProjectIT-*Studio* de forma simples e modular, e que permita ao mesmo tempo que todos esses sistemas colaborem uns com os outros e que possam interagir entre si. Contudo, esta nova

plataforma deve promover o encapsulamento e modularidade dos vários sistemas, seguindo a metáfora de "sistemas que dependem de outros sistemas". Um requisito adicional é a necessidade de esta ferramenta se basear na plataforma .NET [.NET], pois é a plataforma sobre a qual todas as outras ferramentas do ProjectIT-*Studio* estão a ser desenvolvidas. Este requisito está relacionado com o desejo de realizar uma migração rápida e simples dos sistemas (que já existiam) para esta nova ferramenta.

Pretendia-se também o desenvolvimento de um razoável conjunto de módulos que permitam utilizar o melhor possível as potencialidades desta ferramenta, de modo a trazer um maior nível de estabilidade e facilidade de desenvolvimento a todos os sistemas do ProjectIT-*Studio*, entre as quais módulos que permitam desenvolver um modelador gráfico em UML 2 [UML]. Pretendia-se também a migração da ferramenta *XIS-Tool* para esta nova plataforma de integração.

Foi criada a plataforma pretendida, e foram feitas as alterações necessárias para que esta se tornasse numa versão preliminar da ferramenta ProjectIT-*Studio*. Foram convertidos alguns módulos que irão certamente auxiliar o desenvolvimento dos vários sistemas do ProjectIT-*Studio*, nomeadamente o módulo GEF ("*Graphical Editing Framework*") [EclipseGEF], que permite criar módulos para edição gráfica (de modelos ou do que o programador quiser). Foi também migrada a ferramenta *XIS-Tool* para a nova plataforma (embora a ferramenta não tenha sido alvo de quaisquer melhorias significativas), iniciando assim a implementação do sistema ProjectIT-MDD como um módulo[1] na nova ferramenta ProjectIT-*Studio*.

A Figura 1.3, no fim deste capítulo, apresenta um mapa de conceitos, com o propósito de resumir este trabalho e facilitar a sua análise. Através desta figura, percebe-se que o objectivo principal pretendido é fornecer uma nova plataforma para a integração dos vários sistemas do ProjectIT.

Durante a realização deste TFC, para além do trabalho propriamente dito, foram produzidos outros resultados: uma apresentação sobre "Padrões de Desenho na plataforma Eclipse" (realizada na Faculdade de Ciências e Tecnologia da Universidade de Coimbra, a 16 de Março de 2005) [SeminárioDesignPatterns] e um artigo científico apresentado na *First*

International Conference of Innovative Views of .NET Technologies (IVNET'05 – realizada no Porto, a 22 de Junho de 2005) [Saraiva2005].

1.4 Organização do Relatório

Este documento descreve o trabalho desenvolvido, estando estruturado em cinco capítulos. No Capítulo 2, apresenta-se o estado da arte relativamente aos conceitos, técnicas e ferramentas relevantes para este trabalho. No Capítulo 3 é apresentada a arquitectura da plataforma desenvolvida neste trabalho. No Capítulo 4 é apresentada a ferramenta *XIS-Tool* e descrito o processo da sua integração na ferramenta ProjectIT-*Studio*. No Capítulo 5 são apresentadas as conclusões deste trabalho e o trabalho futuro.

O documento é complementado por sete anexos (A-G) que descrevem mais detalhadamente certos aspectos importantes para este trabalho.

[1] Adopta-se neste documento o termo "sistema" como designação da funcionalidade em si (a nível abstracto) e o termo "módulo" como designação da implementação dessa mesma funcionalidade.

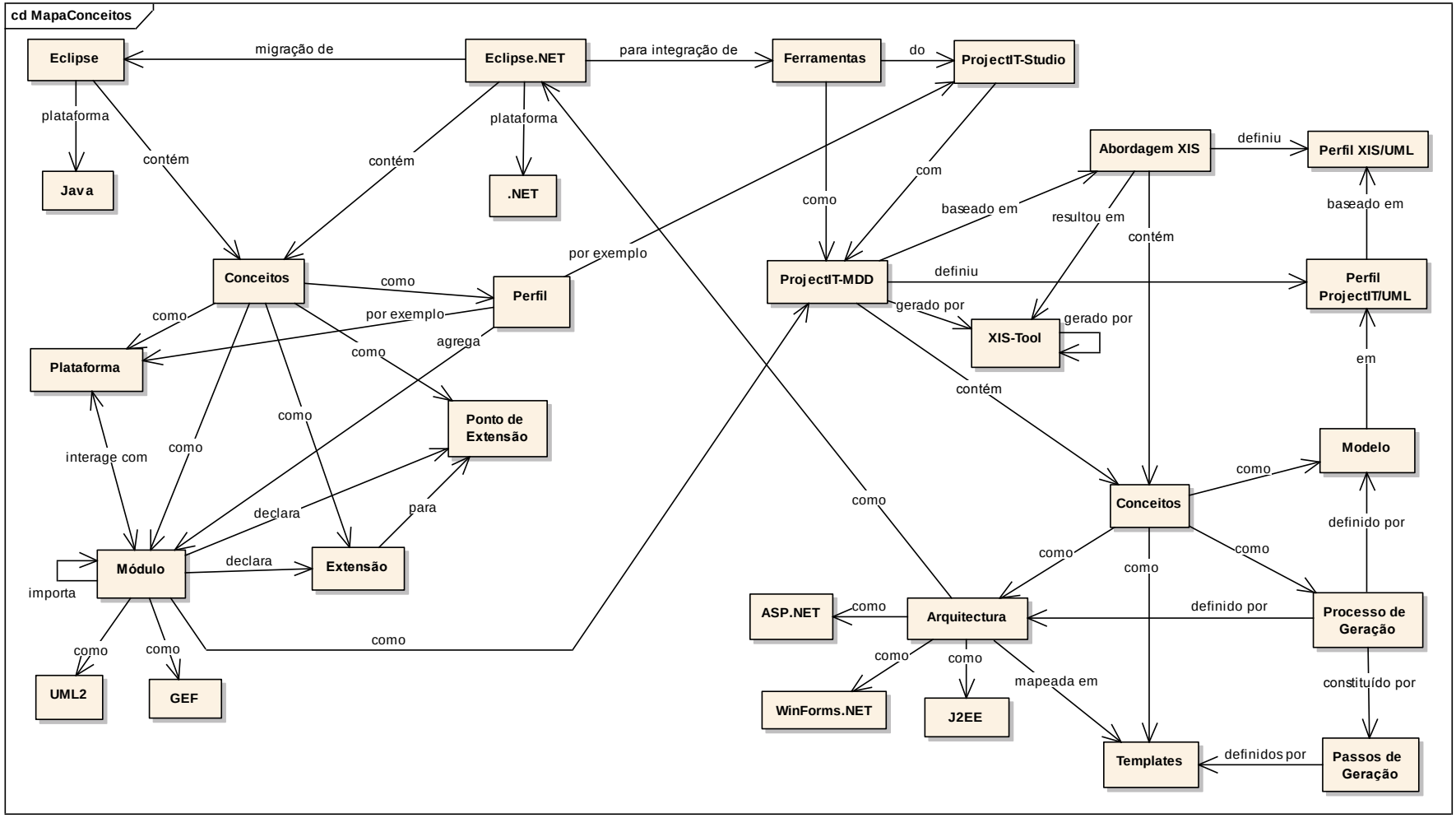

Figura 1.3 – Mapa dos conceitos relevantes deste trabalho.

2 Estado da Arte – Conceitos, Técnicas e Ferramentas

Neste capítulo é feita uma análise/descrição do estado dos vários conceitos, técnicas e ferramentas utilizados no âmbito deste trabalho, nomeadamente: é apresentado o estado actual da plataforma .NET; as ferramentas para integração; e as ferramentas de conversão utilizadas. Finalmente, é apresentado o estado dos vários conceitos sobre os quais o ProjectIT-MDD se baseia.

2.1 Plataforma .NET

A plataforma .NET (designada como "*.NET Framework*") [.NET] é uma plataforma cujo objectivo é o desenvolvimento e execução de aplicações e *Web services*. Esta plataforma é constituída por um mecanismo de execução (designado por CLR – "*Common Language Runtime*") e uma biblioteca de classes (designada como "*.NET Framework class library*") destinadas a suportar o desenvolvimento de *software*. A Figura 2.1 mostra um esquema dos componentes da plataforma .NET.

Figura 2.1 – Componentes da plataforma .NET (extraído de [.NET]).

O CLR gere a execução do código do programa, convertendo as instruções em linguagem MSIL (*Microsoft Intermediate Language*, linguagem intermédia para a qual o programa é compilado) para as respectivas instruções em código máquina. O CLR é também responsável por serviços de base como a gestão de memória dos processos e fios de execução (*threads*), tratamento de excepções, compilação e reflexão, ou ligações remotas, mas assegurando a sua segurança e robustez [.NET].

A gestão do código é fundamental na plataforma .NET, e é por isso que existem os conceitos de código gerido (*managed code*) e código não-gerido (*unmanaged code*). O código gerido é direccionado para o CLR enquanto que o código não-gerido não é direccionado para o CLR. A Figura 2.2 ilustra a arquitectura da plataforma .NET, do ponto de vista da relação entre o CLR e o sistema geral (sistema operativo, aplicações, etc.).

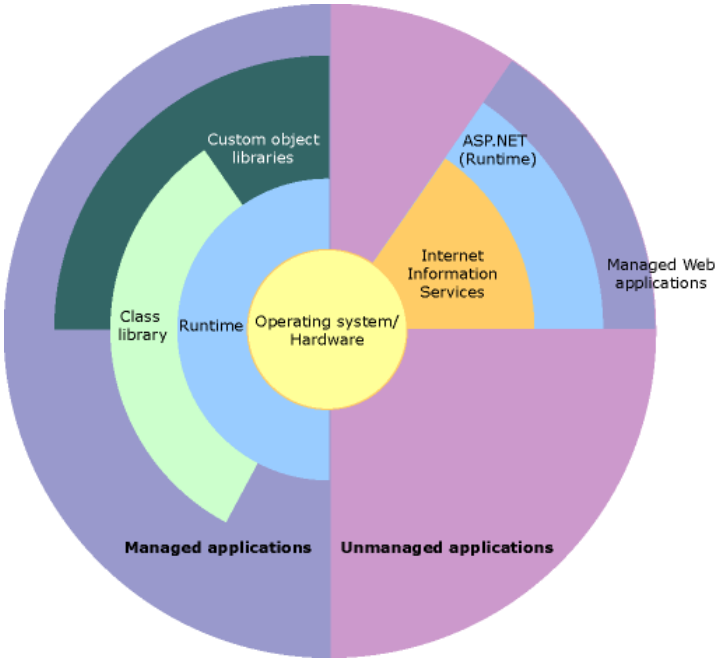

Figura 2.2 – Diagrama da arquitectura da plataforma .NET (extraído de [.NET]).

A biblioteca de classes, a outra componente principal da plataforma .NET, é uma colecção de classes que podem ser usadas para desenvolver aplicações. Estas classes fornecem funcionalidades normais como entrada/saída, manipulação de *strings*, mecanismos de comunicação, criação de interfaces com o utilizador, gestão de colecções, etc.

As aplicações criadas com a plataforma .NET podem ir desde o tipo de aplicações para linha de comandos ou com interface gráfica (GUI – "*Graphical User Interface*") até às aplicações baseadas em ASP.NET [ASP.NET], com "*WebForms*" e "*XML Web services*" [.NET].

Actualmente, a plataforma .NET encontra-se na versão 1.1, com a versão 2.0 planeada para ser lançada no segundo semestre de 2005. Esta nova versão da plataforma irá melhorar a escalabilidade e o desempenho das aplicações. Contudo, a sua maior alteração estará localizada no ASP.NET 2.0 [ASP.NET], com a introdução de uma nova gama de controlos e suporte para mais *browsers* e sistemas computacionais.

2.2 Plataformas para integração

Após um período de pesquisas, concluiu-se que não existia em .NET nenhuma plataforma que facilitasse a integração de todas as ferramentas desenvolvidas no contexto do ProjectIT-*Studio*. Por este motivo, tomou-se a decisão de converter uma plataforma deste género já existente em Java [Java]. Os candidatos a esta conversão são apresentados a seguir.

2.2.1 Plataforma Eclipse

A plataforma Eclipse (também designada apenas por Eclipse) [Eclipse] é uma plataforma em código aberto, conhecida pela comunidade de desenvolvimento Java em geral por permitir o desenvolvimento de *software*, e por oferecer um excelente sistema de extensibilidade (e de comunicação entre extensões).

O Eclipse encontra-se actualmente na versão 3.1.0, com uma recente (e profunda) actualização ao seu núcleo, modificado para se basear na plataforma OSGi (*Open Services Gateway Initiative*) [OSGi], desde a sua versão 3.0.

A plataforma Eclipse consiste numa aplicação sobre a qual se desenvolvem novas extensões. A plataforma trata dos detalhes internos da aplicação (por exemplo: cliques no rato, teclas pressionadas, etc.) e fornece um número considerável de abstracções e bibliotecas de utilitários. A plataforma Eclipse tem também uma enorme comunidade de programadores, que a suportam, produzem documentação e auxiliam na criação e manutenção da plataforma.

2.2.2 Plataforma *NetBeans*

A plataforma *NetBeans* [NetBeans], à semelhança da plataforma Eclipse, consiste numa plataforma para desenvolvimento de novas aplicações. Como na plataforma Eclipse, a plataforma *NetBeans* trata do modelo de eventos internos da aplicação e fornece um número considerável de abstracções, sobre as quais o programador desenvolve o seu código. Tudo o

que o programador tem que fazer é desenvolver módulos que utilizem essas abstracções e comuniquem com a plataforma.

2.2.3 Análise e Comparação das hipóteses

Embora à primeira vista a plataforma *NetBeans* possa parecer tão poderosa como a plataforma Eclipse, esta última apresenta algumas vantagens significativas:

- A comunidade que se baseia na plataforma *NetBeans* é muito menor que a comunidade que se baseia na plataforma Eclipse, o que dificulta o suporte à plataforma *NetBeans*;
- A plataforma Eclipse foi completamente concebida e implementada recorrendo a padrões de desenho [Gamma1995] [Gamma2003], o que leva à menor probabilidade de erros no desenho da plataforma e faz com que a manutenção seja mais simples, para além de tornar mais simples a compreensão da plataforma;
- A plataforma Eclipse trata não apenas da comunicação dos módulos com a plataforma, mas também da comunicação dos módulos entre si. Embora os módulos da plataforma *NetBeans* possam também comunicar entre si, este mecanismo não é de tão simples utilização como é o da plataforma Eclipse.

A seguir, apresenta-se um resumo dos prós e contras de cada plataforma considerada como potencial candidata a conversão, na Tabela 2.1. Após este resumo, apresenta-se e justifica-se a decisão tomada relativamente à ferramenta a ser convertida.

Tendo em conta a comparação feita na tabela anterior, foi tomada a decisão de converter a plataforma Eclipse para .NET. Esta decisão baseou-se no facto de haver uma enorme comunidade envolvida na plataforma Eclipse (o que implica que há muita documentação disponível na Internet, bem como um grande número de extensões já desenvolvidas e testadas). Planeia-se disponibilizar esta conversão como um projecto de código aberto; se esta conversão for bem aceite pela comunidade, a equipa do ProjectIT não se terá que preocupar com o esforço de criação e manutenção da plataforma.

Tabela 2.1 – Prós e Contras das hipóteses para ferramenta de integração

	Prós	Contras

Plataforma *NetBeans*	• Muitas abstracções, sobre as quais o programador desenvolve	• A comunidade não é muito grande
		  ○ Manutenção, suporte à ferramenta e esclarecimento de dúvidas não é fácil de obter
		  ○ Não há tanta documentação disponível na Internet como para a plataforma Eclipse
		• Só existe na plataforma Java
		  ○ Exige esforço de conversão

Plataforma Eclipse	• Modelo de comunicação entre extensões simples e eficaz
	• Comunidade muito grande
	  ○ Bom suporte à ferramenta e esclarecimento de dúvidas
	• Tendo em conta a comunidade já existente, a conversão para .NET pode ter grande aceitação, o que é bom para a equipa do ProjectIT pois:
	  ○ Implica pouco esforço de criação e manutenção da plataforma e respectiva documentação
	  ○ A equipa do ProjectIT pode concentrar-se no seu próprio projecto
		• Pode ser "demasiado completa/complexa" para o problema em questão
		• A curva de aprendizagem é considerável
		• Só existe na plataforma Java
		  ○ Exige esforço de conversão

Contudo, foi tomada a decisão de converter a versão 2.1.3 da plataforma Eclipse (em detrimento da versão mais recente, 3.1.0). Esta decisão deve-se ao facto de a versão 2.1.3 não depender da plataforma OSGi; a conversão dessa plataforma iria demorar bastante tempo e, tendo em conta o limite de tempo imposto para a realização deste TFC, não seria possível converter a plataforma Eclipse e a plataforma OSGi no tempo disponível. Além disso, a versão 2.1.3 oferece praticamente todas as funcionalidades da última versão: a grande actualização entre a versão 2.1.3 e a versão 3.0 foi a modificação do núcleo do Eclipse para se basear na plataforma OSGi.

2.3 IKVM.NET

O projecto IKVM.NET [IKVM.NET] visa produzir uma ferramenta que permita obter total interoperabilidade entre as plataformas Java e .NET. Para atingir este objectivo, existe uma camada intermédia, que recebe as chamadas à máquina virtual Java, e devolve as instruções equivalentes na plataforma .NET. A Figura 2.3 ilustra o modo de funcionamento da ferramenta IKVM.NET e como esta tenta atingir interoperabilidade entre as duas plataformas.

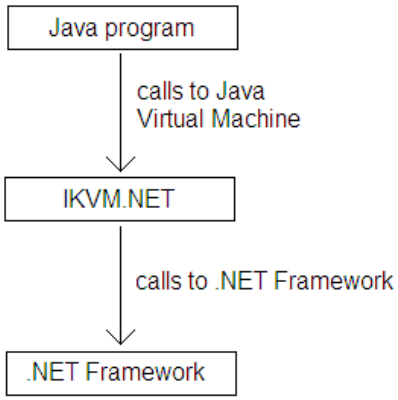

Figura 2.3 – Modo de funcionamento da ferramenta IKVM.NET.

Este projecto está em estado bastante avançado, com muitas aplicações Java a serem executadas correctamente sobre a plataforma .NET. Contudo, a grande utilidade desta ferramenta para este TFC será o conversor de bibliotecas incluído com a ferramenta. Este conversor recebe como parâmetro um conjunto de classes Java (ficheiros *.class*), ou uma

biblioteca com essas classes (um ficheiro *.jar*), e devolve uma biblioteca de classes .NET (um ficheiro *.exe* ou *.dll*, conforme a biblioteca seja executável ou não, respectivamente)[2].

2.4 *Java Language Conversion Assistant*

A ferramenta *Java Language Conversion Assistant* (JLCA) [JLCA] é uma ferramenta que permite a conversão de código e chamadas à API (*Application Programming Interface*) Java para o código e chamadas equivalentes em código C#.

A ferramenta suporta algumas bibliotecas disponíveis na plataforma Java (por exemplo, as chamadas à biblioteca das colecções em Java podem ser convertidas em chamadas às bibliotecas de colecções em .NET). Está estimado que cerca de 80% do código de uma aplicação pode ser convertido automaticamente; o que não for convertido é registado tanto num relatório final como também no próprio código [JLCA][3].

Esta ferramenta é disponibilizada pela Microsoft como uma extensão gratuita ao *Microsoft Visual Studio .NET* [MSVisualStudio] e revelou-se bastante útil na conversão de grande parte do código Java da plataforma Eclipse, código esse bastante volumoso, o que permitiu uma enorme poupança de tempo na conversão.

2.5 Arquitecturas de *Software*

A arquitectura de *software* de um programa ou sistema computacional é a estrutura ou estruturas do sistema onde se incluem os componentes, as propriedades externamente visíveis desses componentes, e as relações entre componentes.

O desenvolvimento de *software* tendo por base uma arquitectura bem definida e documentada facilita a comunicação entre os intervenientes no processo de desenvolvimento. O utilizador tem preocupações de fiabilidade e disponibilidade. O cliente deseja uma implementação nos limites do planeamento e orçamento. O gestor pretende que a equipa trabalhe de forma contínua, independentemente de interacções exteriores, mesmo que disciplinadas e controladas. A equipa de desenvolvimento procura estratégias para atingir

[2] A utilização da ferramenta IKVM.NET no âmbito deste trabalho é descrita com mais pormenor no Anexo A.

[3] A utilização da ferramenta JLCA no âmbito deste trabalho é descrita com mais pormenor no Anexo A.

todos estes objectivos. A arquitectura fornece uma linguagem comum na qual diferentes aspectos podem ser expressos, negociados, e resolvidos a um nível que é manejável intelectualmente mesmo para sistemas grandes e complexos. A formação de novos elementos no projecto é facilitada por existir uma documentação dos principais conceitos arquitecturais e pela regularidade das soluções usadas na implementação, factor relevante em grandes sistemas, pois reduz a impedância da portabilidade do programador.

Recomenda-se, para uma leitura mais extensiva sobre o tema das arquitecturas de *software*, a consulta de [Bass2003] e [Clements2002].

2.6 UML (*Unified Modelling Language*)

O UML (*Unified Modelling Language*) [UML] é uma linguagem diagramática, utilizável para especificação, visualização e documentação de sistemas de informação [UML] [Silva2005b]. O UML permite comunicar de forma *standard* o desenho de uma aplicação, que não só oferece um padrão que serve de guia à implementação, como tem um papel fundamental na documentação das decisões que são tomadas e na compreensão dos potenciais riscos antes da implementação.

O UML encontra-se estruturado numa arquitectura de quatro camadas: meta-meta-modelo, meta-modelo, modelo e objectos do utilizador, que se distinguem pelo nível de generalidade e abstracção dos seus elementos constituintes. Estas quatro camadas são ilustradas na Figura 2.4.

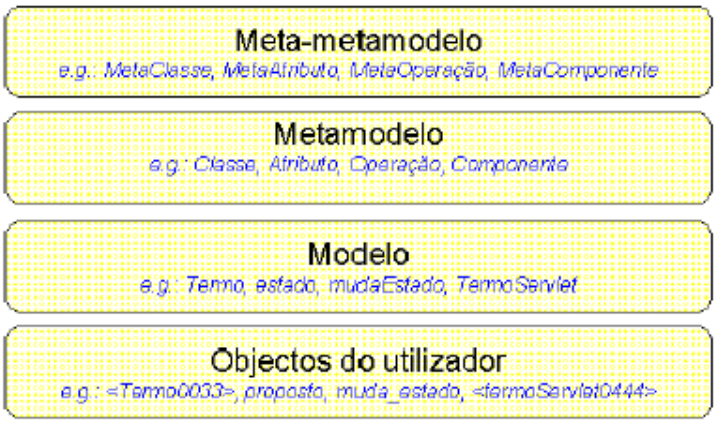

Figura 2.4 – Arquitectura do UML (extraído de [Silva2001]).

Este tipo de arquitectura é adequado para a definição de modelos complexos. Na construção de um modelo UML são criados os elementos da camada de modelo e objectos usando os elementos da camada meta-modelo. Contudo, é possível definir novos elementos do meta-modelo usando os elementos do meta-meta-modelo. O UML oferece alguns mecanismos de extensão que se aplicam a elementos do modelo, representando portanto extensões à própria linguagem. É possível acrescentar informação semântica arbitrária a qualquer elemento usando marcas de valor, ou classificar os elementos usando estereótipos.

Um perfil UML é um conjunto pré-definido de estereótipos, marcas com valor, restrições e ícones que conjuntamente especializam e configuram o UML para um determinado domínio de aplicação ou um determinado processo de desenvolvimento. Contudo, um perfil não estende o UML através da introdução de novos conceitos de base [Silva2005b].

Para informação mais detalhada sobre o UML, recomenda-se a consulta de [UML] e [Silva2005b]. Recomenda-se também a consulta de [Lemos2003] e [Queiroga2004] para uma descrição detalhada da aplicação do UML à ferramenta *XIS-Tool*, na qual o ProjectIT-MDD se baseia (em [Queiroga2004], são também dados detalhes específicos relativamente à adaptação da abordagem XIS ao UML 2.0).

2.7 Linguagens XML

2.7.1 XML (eXtensible Markup Language)

O XML (*eXtensible Markup Language*) [XML] é uma norma para representar e partilhar informação de forma consistente, sendo uma recomendação formal do W3C (*World Wide Web Consortium*) [W3C]. O XML oferece uma forma flexível de criar formatos de informação e partilhar a informação e o formato na Internet. Para tal, são usadas marcas (*markups*) para descrever os dados que um documento contém e são descritas regras para estruturar a informação com marcas embebidas na informação. É uma linguagem muito extensível porque as marcas são ilimitadas e auto-definem-se.

2.7.2 XMI (XML Metadata Interchange)

O XMI (*XML Metadata Interchange*) [XMI] é a norma OMG [OMG] definida para representar meta-dados e os dados correspondentes aos modelos especificados em UML. Para

tal, especifica uma estrutura de representação de modelos UML conforme o meta-modelo UML. O seu objectivo é permitir a utilização de modelos de forma independente das ferramentas de modelação e repositórios a serem utilizados.

Uma consequência desta independência é a possibilidade de partilhar modelos UML pela Internet, em formato XMI, de forma *standard*. Isto permite a utilização desses mesmos modelos em ferramentas e ambientes diferentes.

O XMI combina os benefícios do XML (para definir, validar e partilhar formatos de documentos) com os benefícios do UML (que oferece uma linguagem comum para especificar, visualizar, construir e documentar objectos e modelos de negócio).

2.8 MDA (*Model Driven Architecture*)

O MDA (*Model Driven Architecture*) [MDA] é uma norma proposta pela OMG, que tem como objectivo definir uma abordagem para a especificação de sistemas de informação, sendo esta abordagem baseada no desenvolvimento orientado por modelos (MDD – *Model Driven Development*).

O MDA é suportado pelos conceitos de PIM (*Platform Independent Model*) e PSM (*Platform Specific Model*) [MDA]. O PIM consiste num modelo independente de qualquer plataforma tecnológica, que pode ser mapeado para modelos com detalhes específicos de várias plataformas tecnológicas. Estes últimos modelos são designados por PSM. Isto permite a uma organização mudar a tecnologia que suporta o seu negócio, mantendo as regras e processos de negócio (descritos pelo PIM); para tal, basta mapear o PIM (que não sofrerá alterações) para um PSM relativo à nova tecnologia de suporte ao negócio. Outra ideia fundamental é a de que, a partir do PSM, o código do sistema seja gerado automaticamente [MDA] [Silva2005b].

Para um maior nível de detalhe sobre a abordagem MDA, recomenda-se a consulta de [MDA]. Para um maior nível de detalhe acerca da utilização desta abordagem no contexto da ferramenta *XIS-Tool*, na qual se baseia o ProjectIT-MDD, recomenda-se a consulta de [Lemos2003] e [Queiroga2004].

2.9 Ferramentas CASE (na plataforma .NET)

Uma ferramenta CASE (*Computer Aided Software Engineering*) pode ser definida como "um produto informático destinado a suportar uma ou mais actividades de engenharia de *software*, relacionada com uma (ou mais) metodologia(s) de desenvolvimento" [Silva2001]. O objectivo principal destas ferramentas é fornecer um ambiente integrado onde a produção de *software* seja o mais simples possível, abrangendo todas as etapas do desenvolvimento, desde a sua concepção até à sua instalação.

As ferramentas CASE também têm como objectivo automatizar as actividades de produção. Para isto, os modelos de análise (que reflectem o domínio de negócio) devem ser transformados, de modo iterativo e incremental, em modelos de desenho (que reflectem a implementação da solução no contexto do domínio de negócio). Estes modelos poderão ser transformados em código (esta operação designa-se por "*forward engineering*"), e o código poderá ser também transformado em modelo (esta operação designa-se por "*reverse engineering*"). A combinação destas duas operações (*reverse* e *forward engineering*) designa-se por "*round-trip engineering*". O objectivo máximo das ferramentas CASE é fornecer um ambiente que permita o desenvolvimento de *software* baseado em modelos e que permita a sincronização entre modelos e código. A Figura 2.5 ilustra a relação entre estes conceitos.

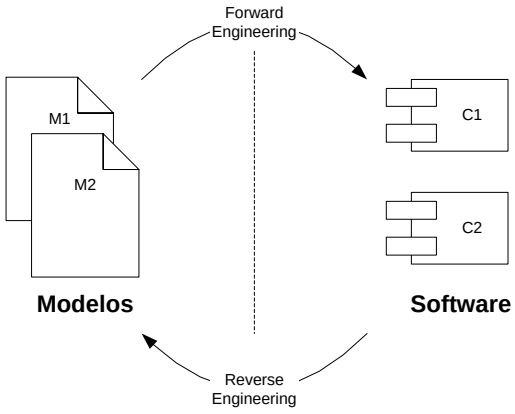

Figura 2.5 – Sincronização entre modelos e código (extraído de [Lemos2003]).

Após um período de pesquisa, concluiu-se que não foram ainda desenvolvidos modeladores gráficos de UML com código aberto e que usem a plataforma .NET. De seguida, apresenta-se algumas das ferramentas CASE que foram analisadas para serem eventualmente utilizadas (e/ou modificadas). No final, é apresentada uma comparação entre as várias soluções.

2.9.1 *Business Change Modeller*

O *Business Change Modeller* é um modelador desenvolvido por José Faria no âmbito de um TFC anterior [Faria2003]. Consiste num editor UML, para a plataforma CEO (Centro de Engenharia Organizacional) [CEO], desenvolvido sobre a plataforma .NET.

O propósito original deste editor era fornecer uma maneira fácil de representar a mudança pela qual os processos de negócio devem passar, ao longo da sua vida. Contudo, o autor seguiu a abordagem de fazer um editor o mais geral possível e posteriormente acrescentar/restringir as funcionalidades disponíveis através de extensões (a título de exemplo: a própria representação da mudança, no editor, é conseguida através de extensões desenvolvidas para o efeito).

Infelizmente este editor ficou bastante incompleto, com uma funcionalidade bastante reduzida (o autor propunha-se a um objectivo muito ambicioso, mas só teve quatro meses para o desenvolver). Isto levou a que o seu mecanismo de extensões tenha ficado num estado muito incipiente. Por causa de todas as limitações encontradas, esta hipótese foi posta de parte.

2.9.2 *MonoUML*

O *MonoUML* [MonoUML] é uma ferramenta CASE para a plataforma Mono [Mono], que é uma implementação da plataforma .NET de código aberto. No início deste trabalho, esta ferramenta ainda se encontrava nas suas primeiras fases de desenvolvimento e, portanto, num estado demasiado imaturo. Por essa razão não foi considerada viável inicialmente. No entanto, não foi posta de parte a hipótese de se utilizar esta ferramenta (ou partes dela) no ProjectIT-*Studio*, num futuro próximo. Aliás, o *MonoUML* cresceu bastante desde a altura em que foi feita a avaliação ao estado da arte desta ferramenta: tem agora uma quantidade considerável de programadores a contribuírem para o seu desenvolvimento e está num estado de desenvolvimento bastante avançado (já permite o desenho de alguns diagramas em UML2, bem como *reverse* e *forward-engineering*, por exemplo).

2.9.3 Ferramentas CASE existentes em Java (*ArgoUML* e *Thorn*)

Colocou-se também a hipótese de converter um modelador existente na plataforma Java. Após alguma pesquisa, os candidatos mais fortes foram o *ArgoUML* [ArgoUML] e o *Thorn* [Thorn]. Ambas as hipóteses foram postas de lado: o *ArgoUML* dependia de um grande número de outras bibliotecas Java, o que dificultaria a sua conversão; o *Thorn* revelou-se demasiado elementar (para além de ainda só suportar alguns dos tipos de diagramas UML necessários ao perfil XIS/UML), para além de ainda ter algumas dependências de outras bibliotecas Java (embora não em tão grande número como no caso do *ArgoUML*). Além disso, durante o decorrer deste trabalho, o desenvolvimento do *Thorn* foi suspenso, por não haver programadores suficientes para continuar a desenvolver a ferramenta.

2.9.4 Módulo UML2 para a plataforma Eclipse

Colocou-se a hipótese de aproveitar o código do módulo UML2 para a plataforma Eclipse [EclipseUML2], para servir de base ao modelador. Este módulo consiste numa implementação do meta-modelo do UML2, implementação essa que pode ser utilizada por outros módulos e que pode ser utilizada para a criação de modelos. Contudo, este módulo depende de outros módulos para a plataforma Eclipse (como o módulo EMF (*Eclipse Modeling Framework*) [EclipseEMF]), o que tornaria a reutilização deste código algo demorada, pois este teria que ser adaptado para se libertar dessas dependências.

Contrariamente ao que seria de esperar, esta hipótese não foi afastada, devido à escolha da plataforma de integração na qual se irá basear o ProjectIT-*Studio*.

2.9.5 Análise e Comparação das hipóteses

A **Erro! A origem da referência não foi encontrada.** apresenta uma listagem dos prós e contras das hipóteses consideradas para desenvolver o modelador.

Desta análise, podem-se tirar algumas conclusões: a hipótese relativa à extensão UML2 para a plataforma Eclipse não foi afastada devido à escolha para a plataforma de integração a desenvolver; caso estas circunstâncias não se aplicassem, a melhor escolha seria uma adaptação da ferramenta *MonoUML*, adaptando o pouco código existente (na altura desta análise) e desenvolvendo a ferramenta, de modo a suportar o perfil ProjectIT/UML.

Tabela 2.2 – Prós e Contras das várias hipóteses para o modelador.

	Prós	Contras
Business Change Modeller	• Desenvolvido em .NET; poupa-se trabalho na conversão do código • O desenvolvimento de extensões é relativamente simples	• Falhas na arquitectura • Mecanismo de extensões limitado • As entidades de base em que o editor se apoia são demasiado pobres para algo como o UML
MonoUML	• Preencheria todos os requisitos • Só seria necessário o desenvolvimento de uma extensão, ou alteração do código fonte • Manutenção grátis	• Demasiado imaturo na altura (encontrava-se em fase de planeamento/codificação inicial)
UML2 (módulo para a plataforma Eclipse)	• Código actualizado • Tendo em conta a escolha para a plataforma de integração, uma adaptação deste módulo pode promover a reutilização de código	• Esforço adicional na conversão • Depende de outros módulos da plataforma Eclipse • Não resolve o problema de ainda ser necessário desenvolver um modelador
ArgoUML	• Muito famoso na comunidade • Bastante poderoso	• Implicaria conversão • Depende de muitas bibliotecas; esforço adicional na conversão
Thorn	• Seria de mais fácil conversão do que o *ArgoUML*	• Mais limitado que o *ArgoUML* • Implicaria conversão • Depende de algumas bibliotecas; esforço adicional na conversão

2.10 Abordagem XIS

A abordagem MDA serviu de "inspiração" para a criação de uma abordagem específica, a abordagem XIS (*eXtensible Interactive Systems*), que tem como objectivo a produção de sistemas de informação interactivos. Na abordagem XIS, assim como na abordagem MDA, a produção de software é centrada em arquitecturas. A Figura 2.6 ilustra um sistema CASE/MDA centrado na arquitectura, no qual se baseia a abordagem XIS.

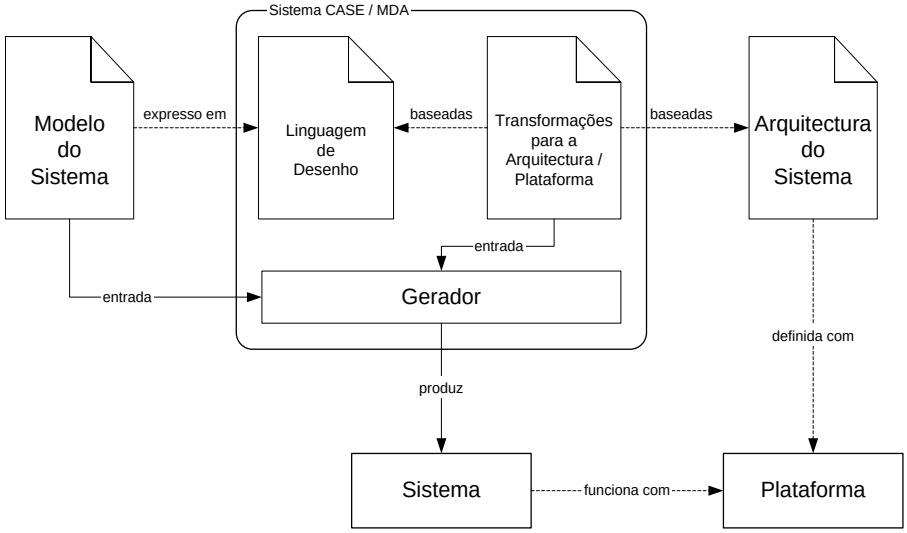

Figura 2.6 – Sistema CASE/MDA centrado na arquitectura (extraído de [Lemos2003]).

A abordagem XIS engloba:

- **Linguagem de Desenho XIS**: esta linguagem permite especificar a informação necessária à construção de sistemas de informação interactivos, de modo a reflectir uma natureza arquitectural no espaço do problema; esta linguagem é concretizada através do Perfil XIS/UML;
    - **Perfil UML para XIS** (ou simplesmente **Perfil XIS/UML**): este perfil consiste num conjunto de extensões UML para a Linguagem XIS, que permite a especificação da informação na Linguagem XIS, de forma intuitiva, visual e de alto nível;

- **Plataforma XIS**: é uma ferramenta cujo objectivo é o suporte e automatização das tarefas dos intervenientes no processo de desenvolvimento (segundo a abordagem XIS), nomeadamente na tarefa de transformação de modelos em código fonte;
    - **Repositório XIS**: este repositório mantém informação sobre modelos, arquitecturas de *software* e o próprio processo de desenvolvimento (nomeadamente, sobre o processo de transformação dos modelos em código).

A Figura 2.7 mostra uma visão geral da abordagem XIS, ilustrando as várias tarefas e as entidades responsáveis pela realização dessas tarefas.

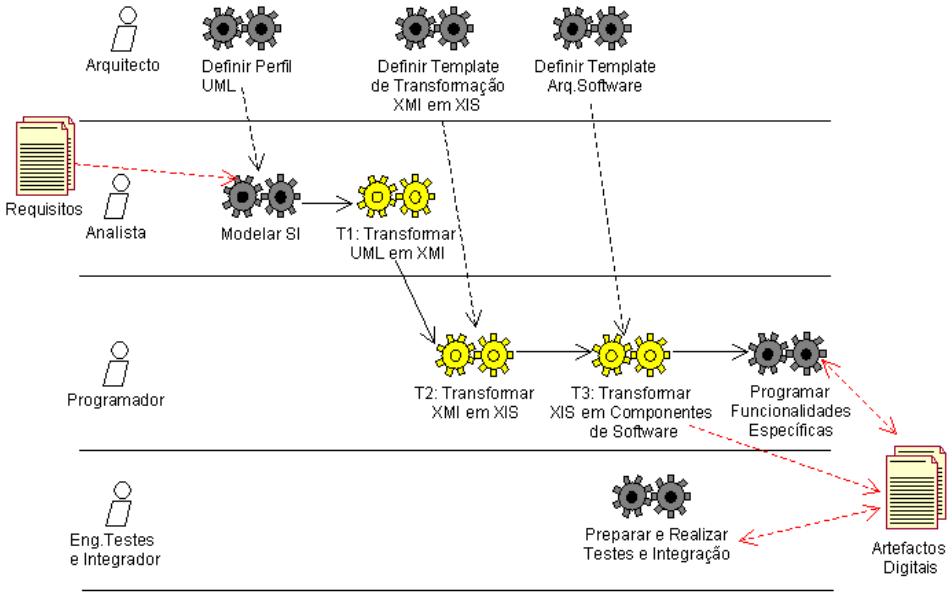

Figura 2.7 – Visão geral da Abordagem XIS (extraído de [Silva2003b]).

Como abordagem de desenvolvimento de software, "a abordagem XIS evidencia alguns dos principais aspectos em discussão no desenvolvimento de novas abordagens ao desenvolvimento de software: (1) baseado em modelos conceptuais abstractos e compactos que representam o sistema; (2) centrada numa arquitectura específica, reflectida e adequada aos requisitos; (3) suportada por técnicas de transformação automática de artefactos digitais" [Silva2003b]. A Figura 2.8 ilustra a concretização da abordagem XIS e a sua inspiração nos conceitos das ferramentas CASE e na abordagem MDA.

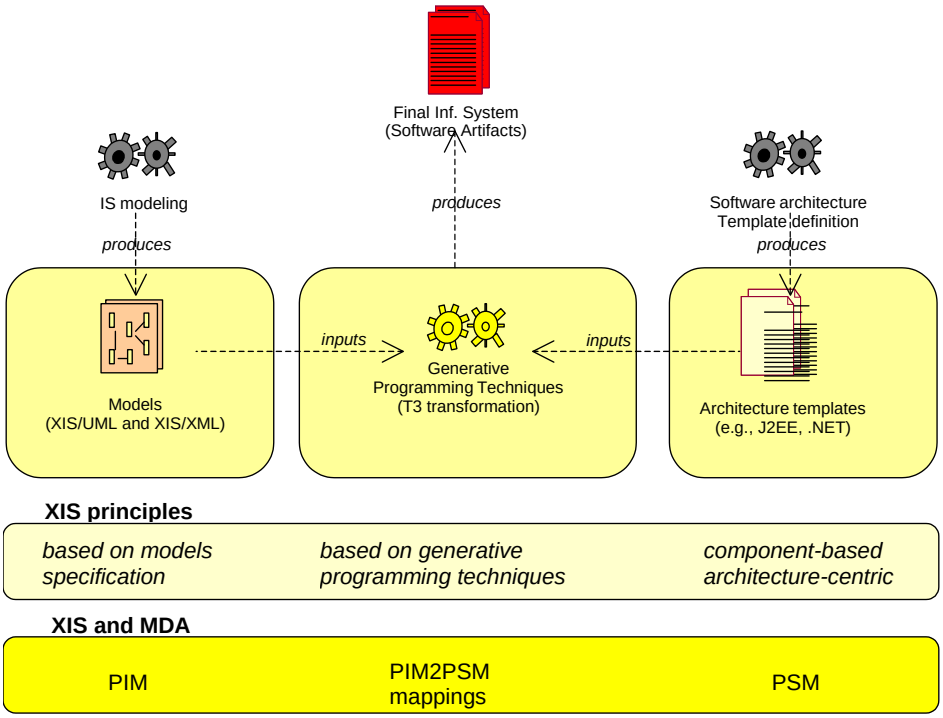

Figura 2.8 – A abordagem XIS, inspirada nas ferramentas CASE e MDA (extraído de [Silva2003a]).

2.10.1 Linguagem de Desenho XIS

A Linguagem de Desenho XIS, que inclui um perfil UML já referido, foi criada para cumprir o objectivo de aumentar a expressividade da linguagem base e "reflectir uma natureza arquitectural no espaço do problema e, consequentemente, no modelo do problema" [Lemos2003]. Relacionada com esta natureza arquitectural está a utilização de um padrão reconhecido para a **organização estrutural** de sistemas de informação interactivos, o MVC (*Model-View-Controller*). É importante não confundir este conceito com o padrão de desenho homónimo. Com esta organização (baseada no padrão MVC), a aplicação interactiva divide-se em três componentes fundamentais: (1) modelo, (2) controlador e (3) vista. "O modelo contém os dados e o núcleo funcional. Os controladores são responsáveis pelo tratamento das interacções com os utilizadores e as vistas realizam a apresentação da informação ao utilizador, sendo que as vistas e os controladores em conjunto constituem a interface com o utilizador" [Silva2003b].

Estes três componentes estão presentes na Linguagem XIS respectivamente como três vistas do problema: *XisModelView*, *XisControlView* e *XisViewView*. Para a descrição dos sistemas segundo a Linguagem XIS, são utilizadas estas três vistas em conjunto com a vista do domínio de negócio, a vista *XisDomainModel*:

- ***XisDomainView***: a vista de domínio pretende definir os objectos que caracterizam o domínio de negócio (***XisEntity***), os seus atributos (***XisEntityAttribute***) e a forma como os objectos se relacionam (***XisAssociation***). É ainda possível especificar relações de generalização/especialização entre os objectos (***XisInheritance***) e listas de valores que definem tipos enumerados (***XisEnumeration***).
- ***XisModelView***: a vista de modelo pretende definir a forma (***XisModelEntity***) como os objectos de domínio (*XisEntity*) são agrupados em entidades de negócio de granularidade superior (***XisModel***), que serão o núcleo funcional isolado da interface com o utilizador.
- ***XisControlView***: a vista de controlo pretende definir os controladores (***XisController***) das interfaces com o utilizador e a sua máquina de estados, especificando estados (***XisState***), transições de estado (***XisTransition***) e acções ou comandos (***XisAction***) que são realizados a partir de eventos despoletados na interface com o utilizador.
- ***XisViewView***: a vista de vistas não pretende definir formas de como é apresentada a informação na interface com o utilizador, mas sim um tipo de vista genérico (***XisView***) que é complementado por um controlador (*XisController*) para realizar a interface com o utilizador de um modelo representativo do núcleo funcional do sistema (*XisModel*).

2.10.2 Plataforma XIS

A plataforma XIS é composta pelos seguintes elementos:

- **Repositório**: o Repositório armazena modelos, processos de geração e arquitecturas;
- **Aplicação de Gestão**: esta aplicação gere a informação contida no Repositório. O utilizador interage com esta aplicação e acede a todo o sistema de geração de código a partir desta;

- **MOXIS** (Modelo de Objectos XIS): este componente corresponde ao modelo de objectos, instanciáveis a partir de modelos persistentes no Repositório, que representam o meta-modelo da Linguagem XIS;
- **Conector**: este componente é responsável por adaptar o modelo UML fornecido (em formato XIS) para o MOXIS; caso o modelo fornecido esteja em formato XMI, este é mapeado para formato XIS e depois adaptado para o MOXIS;
- **Gerador**: este componente é responsável por executar os processos de geração e por escrever o código no sistema de ficheiro;
- **Adaptador**: o adaptador tem como responsabilidade auxiliar no mapeamento entre a informação que se encontra no Repositório e o MOXIS.

A Figura 2.9 mostra uma visão de alto nível da Plataforma XIS, mostrando como todos os elementos apresentados se relacionam entre si.

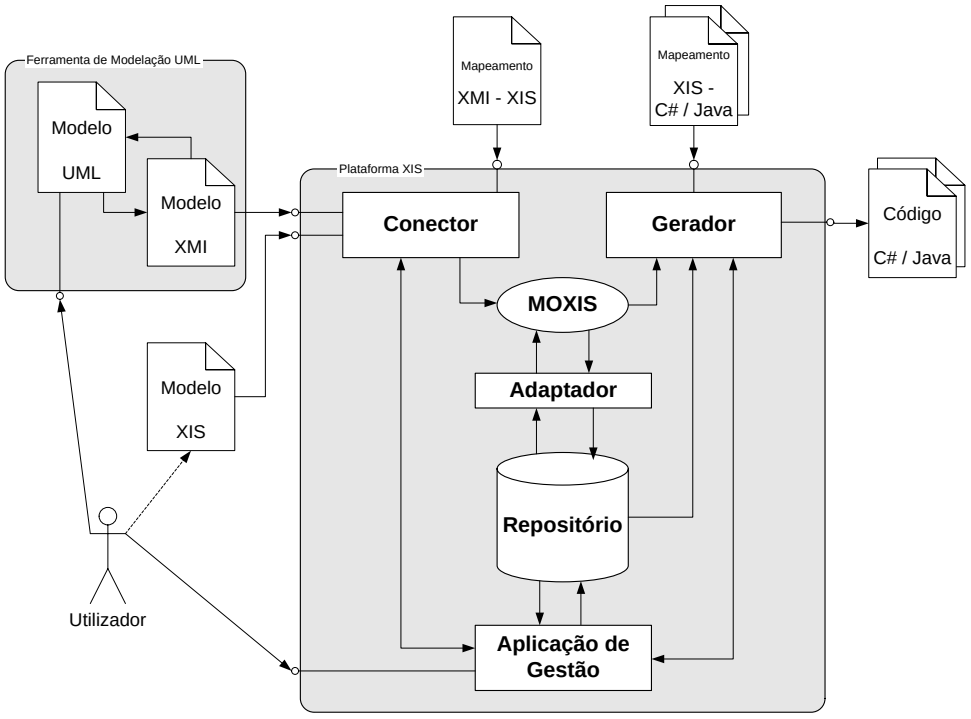

Figura 2.9 – Visão de alto nível da Plataforma XIS (extraído de [Lemos2003]).

O MOXIS (Modelo de Objectos XIS) consiste num conjunto de classes que representam os elementos da Linguagem XIS. Este modelo de objectos pode ser instanciado pelo Conector através de um modelo XIS em XML ou pelo Adaptador (acedendo ao Repositório), estando assim estes objectos isolados dos detalhes de implementação do Repositório ou da gramática do documento XML (seria indesejável ter como entrada a uma aplicação geradora de código um modelo XIS representado em XML: para além do fardo do processamento do documento XML, seria necessário incluir na aplicação os detalhes da gramática que o documento respeita). No caso do sistema de geração de código, as classes do MOXIS são utilizadas nos *templates* de geração de código durante a navegação nas diferentes partes do modelo (dado como entrada ao processo de geração).

2.10.2.1 Geração

A funcionalidade de geração é implementada pela entidade "*Generative Process*" (processo de geração). Um processo de geração (*GenerativeProcess*) agrega um conjunto de etapas de geração (G*enerativeStep*). Uma etapa de geração define o conjunto de transformações, que através de uma arquitectura, são aplicadas a um conjunto de elementos do modelo (*SubSystemModel*).

As arquitecturas (*Architecture*) são identificadas por um nome e compostas por um conjunto de camadas (*ArchitectureLayer*) que reflectem a organização dos elementos de cada arquitectura. A cada camada é igualmente associada uma ou várias transformações de modelo para implementação (*TemplateModule*) que representam os conceitos arquitecturais.

Estes processos são reflexo de uma abordagem iterativa e incremental, pois exprimem a ligação entre subsistemas do modelo e camadas da arquitectura, que serão as entradas no gerador de código. Assim, é possível segmentar a geração através das etapas de geração (tendo em consideração as camadas da arquitectura), permitindo a possibilidade da geração do código relativo a uma camada apenas do sistema ou até só alguns ficheiros, em vez de apenas permitir a geração da aplicação toda de uma vez só. A Figura 2.10 mostra o modelo de classes para processos de geração, ilustrando as relações entre as entidades que integram o processo de geração.

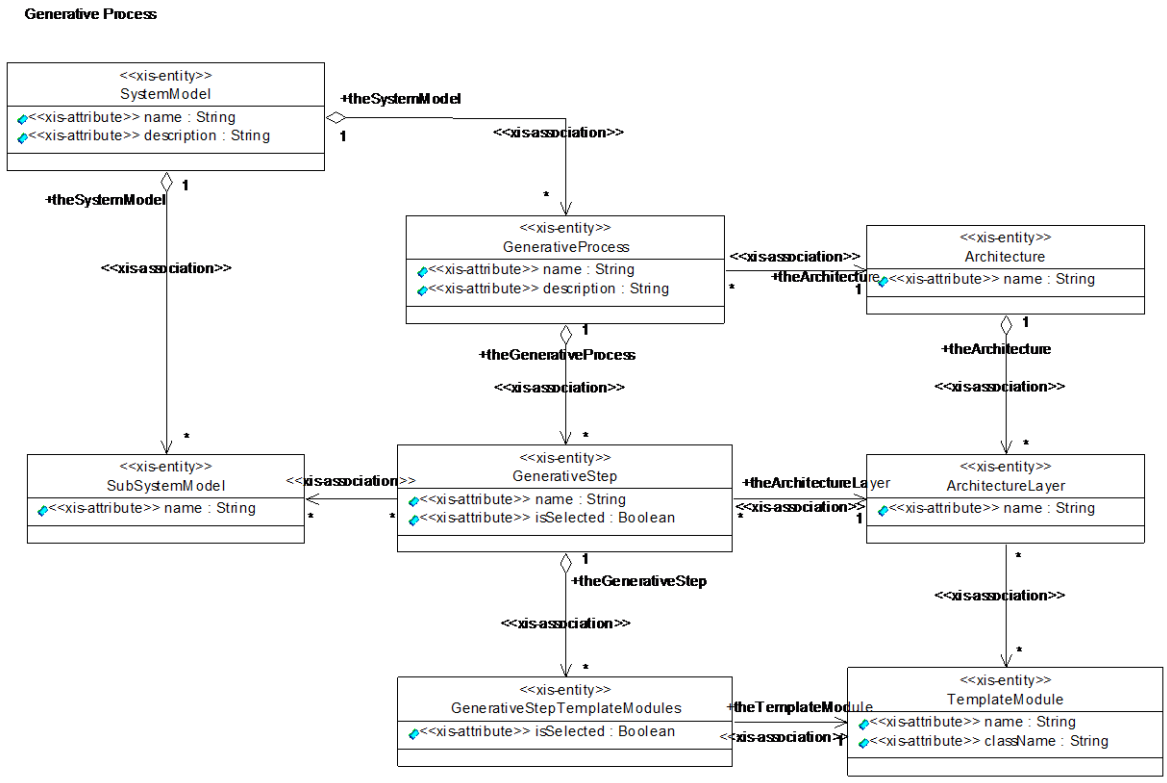

Figura 2.10 – Modelo de classes para processos de geração na Plataforma XIS (extraído de [Lemos2003]).

2.10.2.2 Importação

A importação é o processo de inserção da informação de um modelo (referente ao domínio de um problema) no Repositório, que implementa os elementos da Linguagem XIS (i.e. contém toda a informação do modelo que foi criado na ferramenta de modelação). Esta informação é inserida após duas conversões sequenciais: (1) conversão para um formato XMI (na ferramenta de modelação); (2) conversão para o formato XIS, que é um ficheiro XML resultante de uma transformação do ficheiro XMI, visto este conter informação excedente desnecessária (recomenda-se uma consulta à Figura 2.7 e Figura 2.9 como auxílio à compreensão).

3 Arquitectura da plataforma Eclipse

Neste capítulo, descreve-se a arquitectura da plataforma Eclipse [Eclipse]. São descritos os seus conceitos básicos, nomeadamente: Módulos, Extensões e Pontos de Extensão, Plataforma; é descrito o trabalho realizado na conversão da plataforma, em particular os problemas que surgiram e como foram resolvidos.

Neste documento não se apresenta a plataforma Eclipse em todas as suas vertentes; apenas se apresenta os seus conceitos básicos e como estes podem ajudar na resolução dos problemas constatados relativamente à integração das ferramentas do ProjectIT-*Studio*. Para mais informações (nomeadamente sobre os conceitos a nível da interface gráfica), aconselha-se a consulta de [Eclipse], com especial incidência sobre a parte do "*Technical Overview*".

Finalmente, são descritos no Anexo F os padrões de desenho [Gamma1995] utilizados no contexto da plataforma Eclipse.

3.1 Conceitos básicos

A plataforma Eclipse baseia-se nos seguintes conceitos: **Módulos** (conhecidos como "*Plugins*"); **Extensões** e **Pontos de Extensão** ("*Extension*" e "*Extension Point*", respectivamente); e **Plataforma** (conhecida como "*Platform Runtime*").

Pelo facto de que todos estes conceitos estão fortemente relacionados entre si, não é possível explicar cada um sem mencionar os outros. O conceito de **Perfil** ("*Profile*", ou "*Feature*" na versão original em Java) é também explicado, embora não seja considerado essencial para a compreensão da plataforma.

3.1.1 Módulo (*Plugin*)

Um **módulo** (também designado por "*plugin*") é a unidade mais pequena de funcionalidade da plataforma Eclipse, que pode ser desenvolvida e distribuída separadamente. Normalmente uma ferramenta simples pode ser desenvolvida como um único módulo, enquanto que uma ferramenta mais complexa muito provavelmente terá a sua funcionalidade estruturada em múltiplos módulos. Toda a funcionalidade da plataforma Eclipse está baseada em módulos, incluindo a sua base, conhecida como **Plataforma** (ou "*Platform Runtime*") [Shah2003].

Os módulos são desenvolvidos para a plataforma .NET (i.e., podem ser desenvolvidos em qualquer linguagem que possa ser compilada para MSIL, como C#, *VisualBasic*.NET, *VisualC++*.NET, etc.). Um módulo pode agrupar (1) zero ou mais bibliotecas (ficheiros *.dll*) para a plataforma .NET; (2) alguns ficheiros de configuração do módulo (para leitura apenas); (3) outros recursos (imagens, catálogos, etc.).

Um módulo tem também um **ficheiro de manifesto** ("*Plugin.xml*") que explicita as relações desse módulo com outros módulos. O modelo de ligação de módulos é relativamente simples: um módulo declara um número arbitrário de **Pontos de Extensão** ("*Extension Points*") e um número arbitrário de **Extensões** a um ou mais Pontos de Extensão declarados em outros módulos [Shah2003] (os conceitos de Extensão e Ponto de Extensão são explicados mais à frente neste capítulo).

Um módulo é **activado** quando o seu código, armazenado em uma ou mais bibliotecas, é carregado em memória e as suas instruções de inicialização são executadas. Esta activação deve-se ao facto desse código ser invocado e, portanto, ir ser executado. Até o módulo ser activado, este existe na plataforma Eclipse apenas como uma entrada no Registo de Módulos. O módulo, quando é activado, usa o Registo de Módulos para descobrir e aceder às extensões que foram declaradas (ou "contribuídas") para os seus pontos de extensão. Por exemplo, o módulo que declara o ponto de extensão das preferências do utilizador pode descobrir todas as preferências do utilizador que foram contribuídas e aceder aos seus nomes para construir uma janela de preferências. Isto pode ser feito acedendo apenas ao Registo de Módulos, sem ter que activar nenhum dos módulos contribuintes. Após ser activado, um módulo continua activo até a Plataforma ser desligada (i.e., o utilizador terminar a sua sessão) [Shah2003].

Um ficheiro de manifesto de módulo consiste num documento XML. Um Ponto de Extensão pode também declarar tipos especializados no documento XML, com informação para ser utilizada nas Extensões. Isto permite ao módulo que declara a Extensão fornecer informação ao módulo que declara o Ponto de Extensão correspondente. Para além disso, a informação presente no ficheiro de manifesto está disponível a partir do Registo de Módulos, sem activar o módulo contribuinte. Esta propriedade é fulcral para a plataforma suportar um elevado número de módulos instalados; apenas alguns desses módulos são activados na sessão de utilização. Deste modo, até um módulo ser activado, a sua informação ocupa muito pouco

espaço em memória e o seu impacto no desempenho da inicialização da plataforma é mínimo [Shah2003].

Os módulos podem ser desenvolvidos para diferentes fins, estando apenas limitados pela funcionalidade da plataforma .NET. Temos como exemplo: módulos de suporte gráfico, como o SWT (*Standard Widget Toolkit*) que é uma colecção de utilitários gráficos [EclipseSWT]; o *JFace*, que é construído sobre o SWT e aumenta a quantidade de utilitários fornecidos; e o *WorkBench* que reúne estas duas extensões e fornece o ambiente gráfico do Eclipse.

Um exemplo desta funcionalidade a partir dos módulos consiste nos módulos de suporte gráfico (SWT, *JFace* e *WorkBench*) referidos anteriormente. A Plataforma apenas trata das interacções entre os vários módulos e, em geral, assegura que todos consigam trabalhar em conjunto. Portanto, a Plataforma não apresenta necessariamente uma interface gráfica ao utilizador; é possível que apresente, por exemplo, uma interface apenas textual. São os módulos de suporte gráfico que tratam de fornecer ao utilizador (e ao programador) todo o conjunto de janelas, *wizards*, *widgets*, etc. que estes têm ao seu dispor quando utilizam a plataforma, ou quando desenvolvem novos módulos para esta.

Para além das funcionalidades que os módulos adicionam à plataforma Eclipse, os módulos podem também relacionar-se entre si, através de relações de **importação**. Esta relação permite que um módulo A, que importe um módulo B, consiga utilizar as classes do módulo B, como se essas classes tivessem sido definidas no módulo A. Suponha-se uma situação como a da Figura 3.1, que ilustra alguns módulos ligados por relações de importação (i.e., módulos que importam outros módulos).

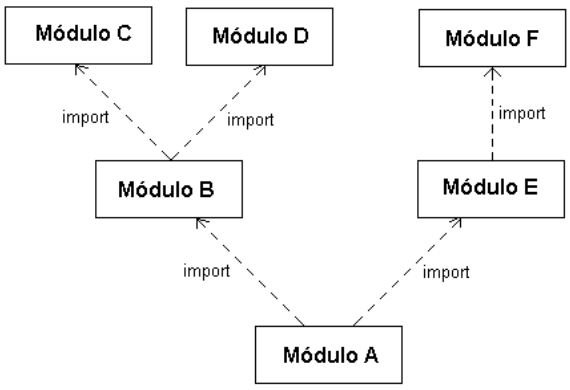

Figura 3.1 – Relações de importação entre módulos.

Se a plataforma tentar activar o módulo A, este módulo e todos os módulos que ele importa são activados recursivamente. Neste exemplo, para se activar o módulo A, activa-se primeiro o módulo B e o módulo E. Mas, para se activar o módulo B, a plataforma terá que activar também os módulos C e D. Como C e D não importam nenhum módulo, estes são apenas activados. Ao activar o módulo E, a plataforma terá que activar o módulo F e só depois irá activar o módulo E. A sequência de acções para a activação do módulo A é apresentada na Tabela 3.1 (note-se que T1 < T2 < ... < T8; além disso, sempre que se indica que não há nenhuma acção executada, é porque esta consiste em adicionar os módulos não-activados à lista de módulos a activar). A Figura 3.2 apresenta essa mesma sequência, mas sob a forma de diagrama de sequência.

Tabela 3.1 – Sequência de activação dos módulos.

Tempo	Módulo X a activar [, outros módulos a activar]	Módulos não-activados de que depende X	Acção executada
T1	A	X = A → { B, E }	–
T2	B, E, A	X = B → { C, D }	–
T3	C, D, B, E, A	X = C → {}	Activar C
T3	D, B, E, A	X = D → {}	Activar D
T4	B, E, A	X = B → {}	Activar B
T5	E, A	X = E → { F }	–
T6	F, E, A	X = F → {}	Activar F
T7	E, A	X = E → {}	Activar E
T8	A	X = A → {}	Activar A

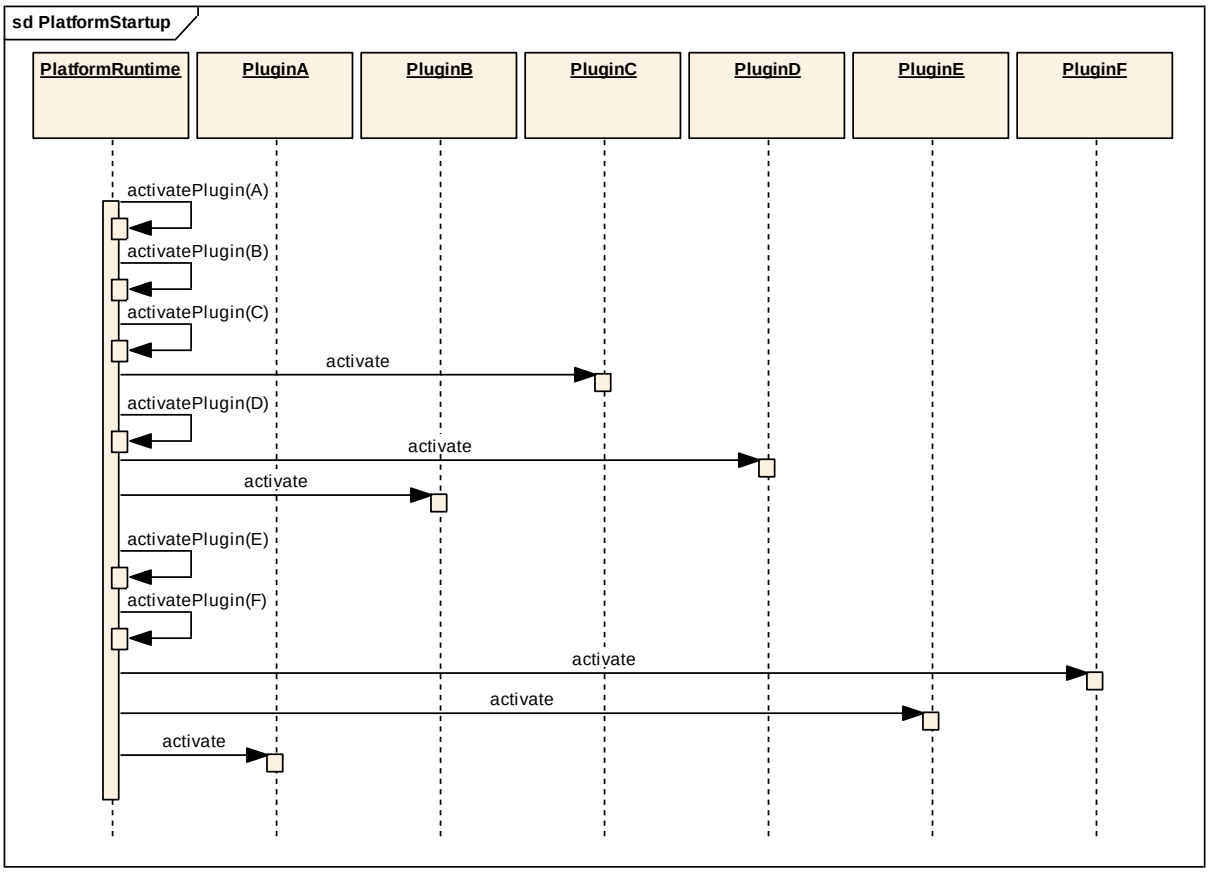

Figura 3.2 – Diagrama de sequência da activação de módulos.

É importante referir que os módulos C e D podem ser activados pela ordem referida, mas também podem ser activados pela ordem inversa (i.e., o módulo D primeiro e depois o módulo C). O mesmo se aplica aos módulos B e E; a ordem apresentada não é relevante. É também importante referir que a sequência de activação apresentada só é concluída caso não ocorra nenhum problema; caso haja algum problema, esta sequência é suspensa e o problema é registado.

Finalmente, é apresentado na Figura 3.3 um exemplo de um ficheiro de manifesto de um módulo (neste caso concreto, o módulo *JFace*).

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
   id="JFace"
   name="%pluginName"
   version="2.1.3"
   provider-name="%providerName">

   <runtime>
      <library name="JFace.dll" />
   </runtime>

   <requires>
      <import plugin="org.eclipse.swt"/>
   </requires>

</plugin>
```

Figura 3.3 – Exemplo de um ficheiro de manifesto de um módulo (*Plugin.xml*).

Neste exemplo, pode-se ver que o módulo *JFace* (na sua versão 2.1.3) contém apenas uma biblioteca (*JFace.dll*), que será carregada aquando da activação do módulo, e que este módulo tem apenas uma relação de importação com o módulo *org.eclipse.swt* (o módulo SWT).

3.1.2 Extensão (*Extension*) e Ponto de Extensão (*Extension Point*)

Outro conceito fundamental é o conceito de **Ponto de Extensão** (*Extension Point*), em conjunto com o conceito de **Extensão** (*Extension*). Os Pontos de Extensão fornecem pontos de variabilidade aos módulos, permitindo que lhes sejam adicionadas funcionalidades.

Os Pontos de Extensão declarados por um módulo podem ser estendidos por outros módulos (i.e., podem ser declaradas Extensões para esses Pontos de Extensão). Por exemplo, o módulo *WorkBench* declara um Ponto de Extensão para preferências do utilizador; para que qualquer módulo contribua as suas próprias preferências do utilizador, este só tem que declarar uma Extensão para este Ponto de Extensão [Shah2003]. A Figura 3.4 ilustra as relações entre módulos e pontos de extensão.

Neste exemplo, o Módulo 1 declara os Pontos de Extensão 1, 2 e 3 (para brevidade do texto, serão designados apenas como PE1, PE2 e PE3, respectivamente). O Módulo 2 declara os Pontos de Extensão 4 e 5. Contudo, o Módulo 2 também contribui para (i.e., fornece Extensões para) PE1 e PE2. Deste modo, o Módulo 1 e o Módulo 2 conseguem comunicar entre si através destes dois pontos de extensão, através dos mecanismos de reflexão disponíveis na plataforma .NET.

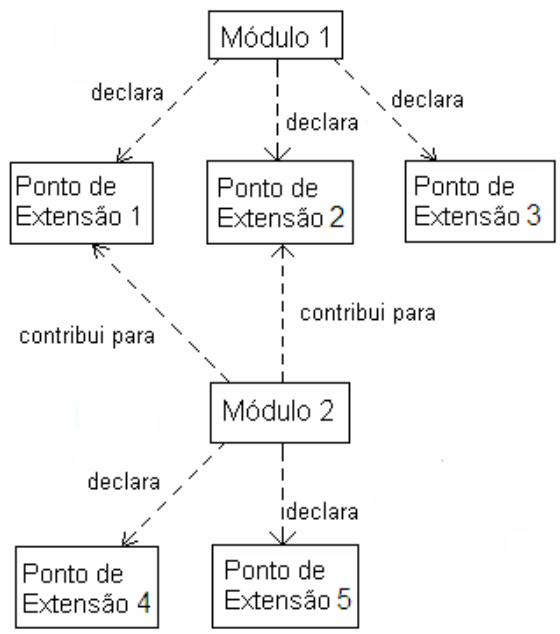

Figura 3.4 – Relações entre Módulos e Pontos de Extensão.

Um facto curioso é que um módulo pode contribuir para Pontos de Extensão que este mesmo tenha declarado. Um exemplo desta situação aconteceria se o Módulo 1, para além de declarar PE1, também contribuísse para esse ponto de extensão. Aliás, é mesmo esta situação que acontece com o módulo *WorkBench*: este módulo declara alguns pontos de extensão, e contribui para alguns desses mesmos pontos de extensão.

Um Ponto de Extensão pode também ter uma API (na forma de uma interface). Outros módulos contribuem com implementações dessa interface, através de Extensões para esse Ponto de Extensão. Qualquer módulo pode definir novos Pontos de Extensão e fornecer novas APIs para serem usadas por outros módulos [Shah2003].

A Plataforma (explicada mais à frente) também declara um Ponto de Extensão especial para aplicações. Quando a Plataforma é iniciada pelo utilizador, o nome da aplicação é especificado como parâmetro; o módulo que é carregado é o módulo que declara a Extensão correspondente à aplicação pedida pelo utilizador [Shah2003].

A Figura 3.5 apresenta um exemplo da declaração de um Ponto de Extensão (neste caso concreto, o Ponto de Extensão para aplicações referido anteriormente). A Figura 3.6 apresenta a declaração de uma Extensão para esse Ponto de Extensão.

```
<extension-point name="%applicationsName" id="Applications"
schema="Schema/Applications.mxsd"/>
```

Figura 3.5 – Declaração de um Ponto de Extensão para aplicações.

```
<extension id="Prompt" point="Core.Runtime.Applications">
   <application>
      <run type="UI.Internal.VersionCheck.VersionCheck" />
   </application>
</extension>
```

Figura 3.6 – Declaração de uma Extensão para o Ponto de Extensão anterior.

No primeiro exemplo (extraído do ficheiro de manifesto do módulo *Core.Runtime*), é declarado um Ponto de Extensão, com identificador "*Applications*". No segundo exemplo, é declarada uma Extensão, com identificador "*Prompt*". Esta Extensão especifica que a classe com o código a ser executado no contexto deste Ponto de Extensão é a classe "*UI.Internal.VersionCheck.VersionCheck*". Chama-se a atenção para o facto que o identificador absoluto de um Ponto de Extensão consiste no nome do módulo que declara o Ponto de Extensão concatenado com '.' e com o nome do Ponto de Extensão no contexto desse módulo (neste caso, "*Core.Runtime*" é o nome do módulo, "*Applications*" é o nome do Ponto de Extensão; o identificador absoluto, utilizado por todos os outros módulos na ferramenta é "*Core.Runtime*" + "." + "*Applications*" → "*Core.Runtime.Applications*"). Quando a aplicação for invocada, irá ser consultado o Ponto de Extensão "*Core.Runtime.Applications*" e, a partir desse Ponto de Extensão, irá ser obtida a Extensão "*Prompt*". A partir dessa extensão, irá ser obtido o nome da classe com o código a ser executado.

Chama-se a atenção para o facto que os tipos "*application*" e "*run*", utilizados no documento XML apresentado na Figura 3.6, são apenas exemplos (neste caso, fazem parte da semântica do Ponto de Extensão "*Applications*"). No caso geral, é da responsabilidade de quem cria o Ponto de Extensão definir também uma semântica para as Extensões a serem contribuídas para esse Ponto de Extensão (definindo os tipos a utilizar, os seus atributos, etc.).

3.1.3 Plataforma (*Platform Runtime*)

A **Plataforma** (também denominada como "*Platform Runtime*") consiste num conjunto de módulos cuja função é iniciar e assegurar o bom funcionamento da ferramenta Eclipse.

Quando o utilizador lança a execução da ferramenta Eclipse, a Plataforma descobre quais os módulos disponíveis, lê os seus ficheiros de manifesto e cria em memória um **Registo de Módulos** (designado "*Plugin Registry*"). A Plataforma também verifica as declarações de extensões com as declarações correspondentes de pontos de extensão. Caso ocorram quaisquer problemas, como extensões contribuídas para pontos de extensão que não existam, estes são detectados e registados. Este Registo está também disponível através da API da Plataforma. Após inicialização da Plataforma, não é possível adicionar-lhe mais módulos de forma dinâmica; caso o utilizador queira adicionar um novo módulo à Plataforma, terá que a reiniciar para que o Registo de Módulos seja reconstruído [Shah2003].

A Plataforma também efectua algumas validações, nomeadamente a nível de pré-requisitos de módulos (relações de importação) e ligações entre extensões e pontos de extensão. Para além de todas estas tarefas, é também responsável por activar os módulos dinamicamente quando estes são necessários.

3.1.4 Perfil (*Profile*)

Um Perfil (designado por "*Profile*", ou por "*Feature*" na versão original em Java) consiste numa abstracção para agrupar módulos, de modo a personalizar a plataforma (i.e., escolher os módulos que farão parte da plataforma, escolher as imagens de logótipo, especificar o nome da plataforma – por exemplo: Eclipse.NET ou ProjectIT-Studio –, etc.). É prática comum criar também um módulo – com a mesma designação do Perfil – que não terá bibliotecas (ficheiros *.dll*), mas que terá apenas imagens pertencentes ao Perfil (por exemplo, logótipos e ícones) e alguns ficheiros auxiliares (como um ficheiro HTML com um texto de introdução ao Perfil – quais os seus propósitos, como o utilizar, etc.).

Após a criação do Perfil, é necessário alterar o ficheiro "*Install.xml*" (no directório de raiz da aplicação que inicializa a plataforma) para reflectir qual o Perfil que a Plataforma deve carregar. Assim, pode haver vários Perfis criados (por exemplo, o Perfil de omissão à Plataforma – designado por "*Platform*" – e o Perfil da ferramenta ProjectIT-Studio – designado por "*ProjectIT.Studio*").

3.1.5 Dos conceitos à plataforma

A Figura 3.7 apresenta um diagrama de alto nível com os conceitos básicos referidos.

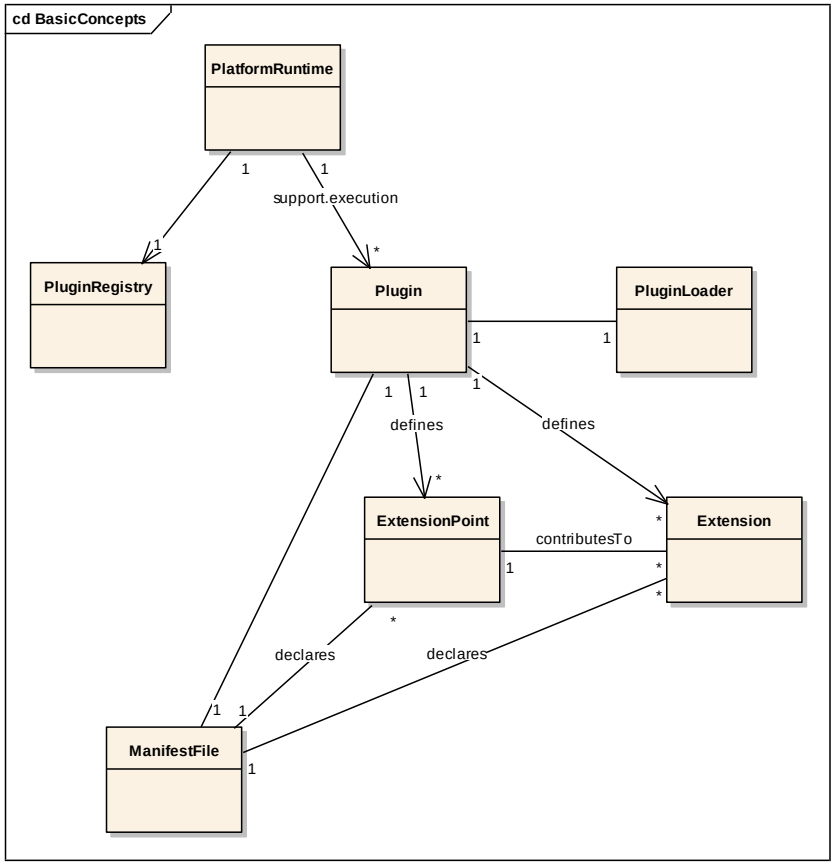

Figura 3.7 – Diagrama de alto nível dos conceitos da plataforma.

A Figura 3.8 apresenta a arquitectura da plataforma Eclipse, numa visão por camadas, com os vários módulos da plataforma Eclipse (recomenda-se a consulta de [Eclipse], nomeadamente o *Technical Overview*, para uma descrição mais detalhada das funções desempenhadas por cada módulo referido na figura). Esta figura apresenta também o conceito de ponto de extensão (e extensões) à plataforma.

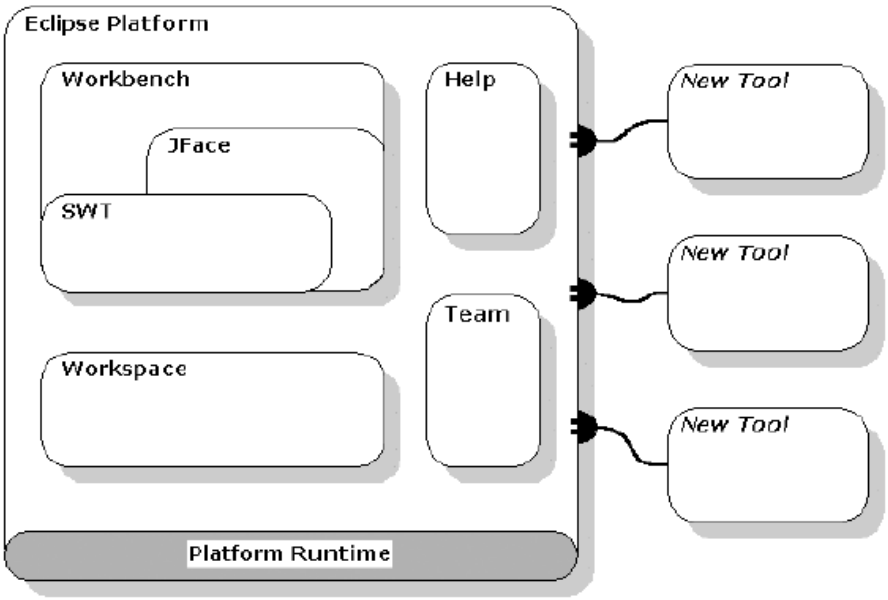

Figura 3.8 – Arquitectura (em vista por camadas) da plataforma Eclipse (extraído de [Eclipse]).

No topo da camada da Plataforma ("*Platform Runtime*") existem vários módulos, entre os quais os de suporte gráfico (SWT – *Standard Widget Toolkit*, *JFace* e *WorkBench*). Note-se que a camada do *JFace* encontra-se sobre a camada do SWT, mas não a encobre totalmente: o módulo *JFace* apenas fornece mais utilitários para a interface gráfica; contudo, o programador tem toda a liberdade para utilizar também o módulo do SWT. É isto que acontece com o módulo *WorkBench*: este é o módulo responsável por fornecer à ferramenta Eclipse a sua interface gráfica, e baseia-se nos módulos SWT e *JFace* para aumentar o seu nível de abstracção. Os módulos "*New Tool*" consistem em novos módulos (novas ferramentas) que o programador pode adicionar à plataforma Eclipse, através do mecanismo de Extensões e Pontos de Extensão.

3.2 Conversão da plataforma

Nesta secção, são apresentados os principais problemas que tiveram que ser resolvidos no decorrer da conversão da ferramenta Eclipse, e o modo como estes foram resolvidos.

É importante referir que a arquitectura da ferramenta, na conversão da plataforma Java para a plataforma .NET, não foi alterada. Isto deveu-se ao facto de uma quantidade muito

significativa do código consistir na aplicação de padrões de desenho (independentes da plataforma de implementação). Isto faz com que a arquitectura (e o código) consigam ser mais facilmente perceptíveis para quem esteja a olhar para estes aspectos pela primeira vez.

A metodologia usada para esta conversão foi a seguinte (as quatro primeiras etapas são descritas e explicadas mais detalhadamente no Anexo A, que descreve as ferramentas utilizadas; as três primeiras etapas foram a conversão ao nível de "*bytecode*" – Java para MSIL – e as duas últimas etapas foram a conversão ao nível do código fonte):

1. Gerar adaptadores ("*stubs*") da plataforma .NET para Java, com a ferramenta IKVM.NET (com estes adaptadores, torna-se possível criar código Java que invoque a plataforma .NET);
2. Com os resultados da etapa anterior, alterar o código fonte (em Java) do módulo SWT para utilizar algumas funcionalidades da plataforma .NET, nomeadamente substituir a utilização de fios de execução (*threads*) Java por fios de execução .NET; compilar o código alterado para classes Java (em vários ficheiros *.class*);
3. Converter para MSIL (e para uma biblioteca *.dll*) as classes obtidas na etapa anterior, com a ferramenta IKVM.NET. No fim desta etapa, o módulo SWT está completamente convertido para a plataforma .NET na biblioteca "*swt.dll*";
4. Com o auxílio da ferramenta JLCA, converter todo o código fonte da plataforma Eclipse (excepto o código do módulo SWT, que foi convertido para .NET na etapa anterior);
5. Corrigir o código de cada um dos módulos convertidos na etapa anterior (tendo em conta as alterações que se fizeram à plataforma, descritas a seguir).

O módulo SWT foi convertido da forma referida acima porque não tem quaisquer dependências da plataforma Eclipse ou do seu mecanismo de "*ClassLoaders*". Caso tivesse alguma dependência deste tipo, seria preciso algum esforço de conversão manual inicial e só depois se poderia converter este módulo.

É de referir que o motivo pelo qual a quase totalidade dos módulos não foi convertida recorrendo à ferramenta IKVM.NET é o desejo de promover o mais possível o alinhamento da plataforma Eclipse.NET (designação dada à plataforma convertida para .NET) com as boas

práticas e funcionalidades que a plataforma .NET oferece. Por outro lado, o módulo SWT é bastante complexo e extenso, e a sua conversão iria demorar demasiado tempo para que esta conversão fosse viável no tempo disponível para a realização deste trabalho, razão pela qual a conversão deste módulo foi feita com a ferramenta IKVM.NET.

Os maiores problemas encontrados não foram ao nível conceptual mas sim a um nível tecnológico (devido às diferenças entre a API da linguagem Java e a API da linguagem C#). Embora a ferramenta JLCA tenha conseguido converter grande parte do código da plataforma, houve algumas falhas na conversão, falhas essas que tiveram que ser corrigidas manualmente.

3.2.1 Mecanismo de carregamento de classes

Um dos problemas encontrados na conversão desta plataforma foi o facto da implementação da plataforma Eclipse se apoiar bastante no mecanismo de carregamento dinâmico de classes (com base em "*Class Loaders*") da plataforma Java [McGraw1999].

Na plataforma Java, este mecanismo permite ao programador especificar a localização de um conjunto de classes (numa directoria, na Internet, etc.) e carregar o "*byte-code*" Java (código intermédio da plataforma Java, à semelhança do MSIL) referente a essas classes. A plataforma Java fornece alguns Carregadores de Classe, como é apresentado na Figura 3.9, e permite ao programador criar os seus próprios Carregadores.

Os Carregadores de Classes têm uma estrutura em árvore, com cada Carregador a ter um Carregador-pai, à excepção do Carregador "*Primordial Class Loader*" (também designado "*BootStrap Class Loader*"). Isto permite que, quando o programador pede uma classe a um Carregador específico, o Carregador procura no conjunto de classes que ele próprio carregou pela classe pedida; caso não a encontre, pede essa classe ao Carregador-pai. Este processo funciona de modo recursivo, tendo como caso terminal o "*Primordial Class Loader*" que, caso não encontre a classe no conjunto de classes que carregou, não devolve nenhuma classe. O Carregador "*java.net.URLClassLoader*" serve para carregar classes Java e recebe como entrada um URL ("*Uniform Resource Locator*") com a localização da classe a carregar.

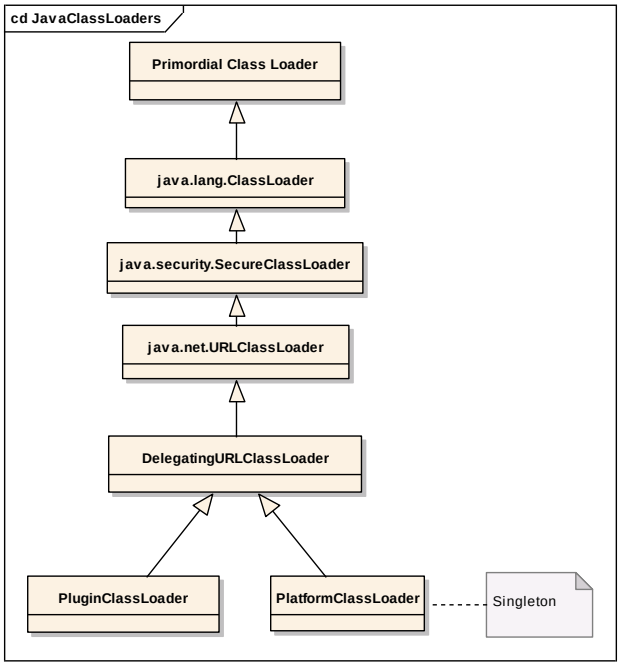

Figura 3.9 – Carregadores de Classes da plataforma Java e na plataforma Eclipse.

A plataforma Eclipse utiliza estes Carregadores de Classes directamente, criando: (1) uma classe abstracta "*DelegatingURLClassLoader*", que herda de "*URLClassLoader*" e que serve para procurar uma classe pedida tanto no próprio "*DelegatingURLLoader*" como nos Carregadores dos módulos que são importados; (2) uma classe "*PlatformClassLoader*", que herda de "*DelegatingURLClassLoader*", que será o Carregador da Plataforma Eclipse e que, por poder haver apenas uma instância da Plataforma carregada, é um "*Singleton*"; e (3) uma classe "*PluginClassLoader*", que também herda de "*DelegatingURLClassLoader*", e que terá uma instância por cada módulo que esteja presente aquando da inicialização da ferramenta; é depois esta classe que trata de carregar os ficheiros *.class* do módulo quando estes são necessários.

Na plataforma .NET este mecanismo não existe. Em vez disso, é utilizado o mecanismo "*Fusion*" [Renaud2003] [Gunnerson2002], que consiste num mecanismo de "*assemblies*" (bibliotecas de código MSIL) [Archer2001] [.NET], que são carregadas para um Domínio de Aplicação ("*Application Domain*" ou "*AppDomain*"). Um Domínio de Aplicação pode ser considerado como um processo lógico (ou seja, um "processo" que existe dentro do processo real no qual se executa a aplicação desenvolvida) ou, como é referido em [Archer2001], "um

processo lógico dentro de um processo físico". Contudo, o programador não tem qualquer controlo sobre o carregamento dessas classes; este carregamento é transparente para o programador, este só tem que especificar os detalhes da biblioteca a carregar (localização, versão, chave pública a utilizar caso necessário, etc.).

Também não existe nada no mecanismo "*Fusion*" para o qual se possa mapear o facto de a plataforma Java poder ter múltiplas instâncias de "*PluginClassLoader*". Para substituir este mecanismo, foram criadas as classes "*Loader*", "*AssemblyLoader*", "*PlatformLoader*" e "*PluginLoader*". A classe "*AssemblyLoader*" esconde os detalhes do carregamento de bibliotecas ("*assemblies*"), recebendo apenas a localização da biblioteca a carregar, e disponibiliza na API métodos para:

- Carregar bibliotecas (e adicioná-las a uma lista interna de bibliotecas carregadas por esse "*AssemblyLoader*")
- Procurar uma classe (por nome) nas bibliotecas carregadas. Este método consiste em procurar (através do mecanismo de reflexão) em todas as bibliotecas carregadas pela classe pedida. Caso esta classe não exista nas bibliotecas carregadas, é pedido à instância de "*Loader*" que procure pela classe. Caso esta devolva um erro (por não encontrar a classe), o método devolve um erro.

A classe "*Loader*" é um "*Singleton*", que contém uma lista de todas as instâncias de "*AssemblyLoader*" criadas e apresenta na sua API um método para que cada "*AssemblyLoader*" se registe consigo (aquando da criação do "*AssemblyLoader*") bem como um método para procurar uma classe por nome. Este método procura em todas as instâncias de "*AssemblyLoader*" existentes pela classe pedida (esta procura é pedida a essas instâncias a partir da API disponibilizada). No caso em que nenhuma instância contenha a classe, o método vai ainda procurar no Domínio da Aplicação (para o caso da classe pedida ser alguma das classes disponibilizadas pelo CLR). Caso o Domínio da Aplicação não contenha a classe, o método devolve um erro. Para efeitos de optimização da procura de classes, a classe "*Loader*" apresenta também um método para devolver o "*AssemblyLoader*" no qual se encontra uma certa classe; isto pode permitir que futuras procuras por classes que estejam dentro desse "*AssemblyLoader*" possam ser mais rápidas.

Após a criação destas classes, a conversão das classes "*PluginClassLoader*" e "*PlatformClassLoader*" foi relativamente trivial e consistiu em fazer o mapeamento entre (1) as classes "*PluginClassLoader*" e "*PluginLoader*"; e (2) as classes "*PlatformClassLoader*" e "*PlatformLoader*". Como na classe "*PluginClassLoader*", é criada uma instância de "*PluginLoader*" por módulo.

Deve-se notar que este mecanismo de "*AssemblyLoaders*" não apresenta nenhuma estrutura em árvore (para simplicidade), ao contrário do mecanismo de "*Class Loaders*" da plataforma Java. É para compensar este facto que é pedido à instância de "*Loader*" que procure em todos os "*AssemblyLoaders*" existentes pela classe pedida (e, em último caso, que a procure também no Domínio da Aplicação).

A Figura 3.10 ilustra as entidades envolvidas neste mecanismo de "*AssemblyLoaders*".

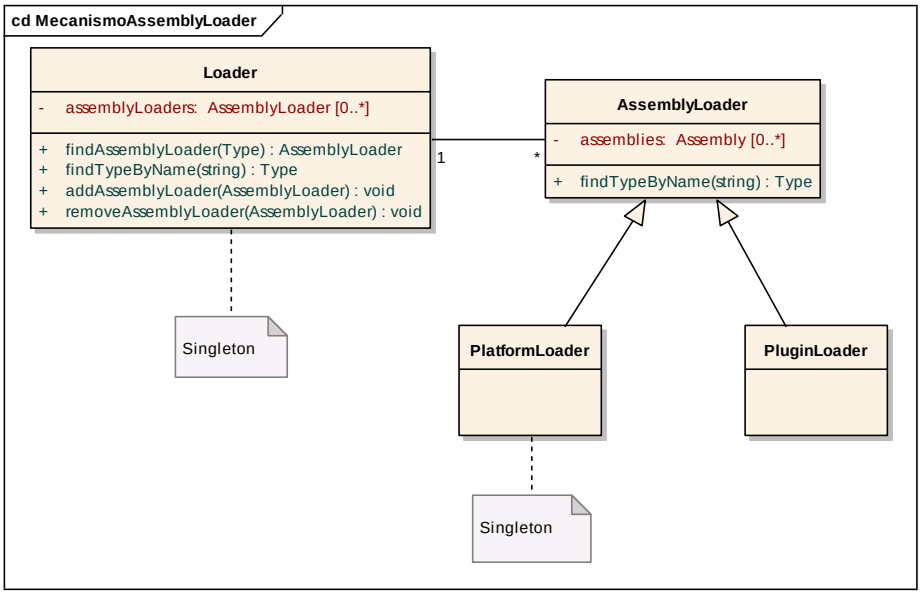

Figura 3.10 – Entidades envolvidas no mecanismo de "*AssemblyLoaders*".

3.2.2 *Getters*/*Setters* vs. Propriedades

Outro aspecto que é importante realçar foi a utilização de Propriedades [Archer2001] quando tal fosse possível, em detrimento de métodos *Getter*/*Setter* (métodos de leitura e escrita em objectos), com o objectivo de alinhar mais esta conversão com a plataforma .NET.

Isto levou a que a API para criação de módulos para esta nova ferramenta tenha sido ligeiramente alterada (com a alteração dos métodos *Getter*/*Setter* para Propriedades). Contudo, a metodologia para criação de módulos continua igual à versão original em Java.

3.2.3 Utilização da ferramenta *Log4NET*

Modificou-se o código da ferramenta desenvolvida para utilizar a ferramenta Log4NET [Log4NET]. O objectivo desta alteração foi um maior controlo na escrita de mensagens (tanto informativas como de depuração), em detrimento de apenas escrever mensagens na consola de saída ("*Console.Out*") e na consola de erro ("*Console.Error*"), que é o que a plataforma Eclipse faz.

Esta ferramenta permite ao programador escrever mensagens para auxílio à depuração, de modo controlado. Caso o programador (ou o utilizador) queira ligar ou desligar esta funcionalidade de registo das mensagens, pode fazê-lo durante a execução da plataforma, sem modificação do código (tanto do código fonte como do código MSIL). Isto faz com que o desempenho da plataforma seja afectado (pela escrita destas mensagens) apenas se tal for necessário.

Esta ferramenta contém também o conceito de "hierarquia de mensagens", o que permite escolher os tipos de mensagens que são mostrados (i.e., se apenas são mostradas mensagens de erro, se são mostradas mensagens de erro e de depuração, etc.). Com esta funcionalidade, pode-se controlar a quantidade de mensagens mostradas, facilitando a depuração da plataforma.

3.2.4 Discussão sobre a conversão entre Java e .NET

De um modo geral considera-se que a metodologia e as ferramentas utilizadas foram adequadas, tendo em conta o tempo disponível para criar uma versão funcional da plataforma Eclipse.NET. Com a ferramenta IKVM.NET, conseguiu-se poupar bastante tempo nesta conversão (pois permitiu obter rapidamente a conversão do módulo que serve de base para o motor gráfico da plataforma). Por outro lado, a ferramenta JLCA permitiu gerar uma grande parte do código (cerca de 90%), e só não conseguiu corrigir alguns problemas (descritos com maior detalhe no Anexo A).

Contudo, uma falha na metodologia (que só foi apontada já durante a conversão) foi o facto de que a ferramenta IKVM.NET poderia ter sido utilizada com mais frequência na conversão, nomeadamente podia ter sido utilizada para apoio à conversão dos módulos de suporte gráfico (*JFace* e *WorkBench*); para tal, a metodologia a seguir teria sido: (1) converter a base da plataforma (i.e., os módulos "*Core*"); (2) gerar adaptadores das bibliotecas geradas por essa conversão; (3) alterar o código Java dos módulos *JFace* e *WorkBench* para acederem às funcionalidades da biblioteca gerada anteriormente, em vez de acederem à versão original (em Java) da plataforma; (4) compilar o código alterado, para classes Java (*.class*); e (5) utilizar a ferramenta IKVM.NET para converter as classes Java obtidas para um biblioteca .NET. Esta metodologia poderia permitir poupar algum tempo na conversão.

Este esforço de conversão também permitiu tirar algumas conclusões sobre o esforço necessário para converter uma aplicação (genérica, não necessariamente a plataforma Eclipse) de Java para .NET. Nomeadamente, concluiu-se que a interoperabilidade entre as plataformas Java e .NET (i.e., aplicações desenvolvidas para uma plataforma a serem executadas sem problemas na outra plataforma) é um assunto que está a ganhar cada vez mais relevância no seio das suas respectivas comunidades.

A ferramenta IKVM.NET revelou-se como excelente auxílio nesta área, com várias aplicações Java (i.e., com ficheiros *.class*) a serem correctamente executadas na plataforma .NET. Contudo, esta ferramenta não promove o alinhamento das aplicações com a plataforma .NET, o que faz com que as aplicações não usem todas as funcionalidades disponíveis na plataforma (como por exemplo a facilidade de utilização de *WebServices*). Em resumo, esta ferramenta é excelente para se passar rapidamente de uma aplicação Java para uma aplicação executável em ambiente .NET, mas não promove o alinhamento da aplicação com esta última plataforma.

A ferramenta JLCA, por outro lado, não converte uma aplicação Java existente (i.e., não converte ficheiros *.class*); em vez disso, converte o seu código fonte para código C#. Isto permite que, durante a transformação desse código fonte, ocorram algumas transformações de modo a promover o alinhamento da aplicação com a plataforma .NET.

Portanto, as ferramentas a utilizar na conversão de uma aplicação (genérica) em Java para .NET dependem dos objectivos que essa aplicação deve preencher: caso a aplicação deva

estar o mais possível alinhada com a plataforma .NET, então a melhor escolha será a ferramenta JLCA, mais a consequente correcção de eventuais falhas que ocorram durante a conversão do código; caso o objectivo seja obter o mais rapidamente possível uma aplicação .NET a partir de uma aplicação Java (sem preocupações sobre o alinhamento da aplicação com a plataforma .NET), a melhor escolha será a ferramenta IKVM.NET. Caso os objectivos sejam uma mistura destes dois (como foi o caso desta conversão, em que o objectivo era criar uma plataforma alinhada com a plataforma .NET, mas não perdendo tempo com módulos que não poderiam ser alinhados com esta última), poder-se-ão utilizar estas duas ferramentas em conjunto.

4 ProjectIT-*Studio*

Nesta secção, descreve-se o processo de desenvolvimento da ferramenta ProjectIT-*Studio* sobre a plataforma Eclipse.NET – designação do resultado do processo de migração da plataforma Eclipse para .NET. Descreve-se também a adaptação da ferramenta *XIS-Tool* para um Módulo Eclipse.NET.

4.1 Desenvolvimento do ProjectIT-*Studio* sobre o Eclipse.NET

Planeia-se desenvolver o ProjectIT-Studio e as suas ferramentas sobre a plataforma Eclipse.NET, como ilustrado na Figura 4.1, que apresenta uma visão por camadas da ferramenta ProjectIT-*Studio* sobre a plataforma Eclipse.NET.

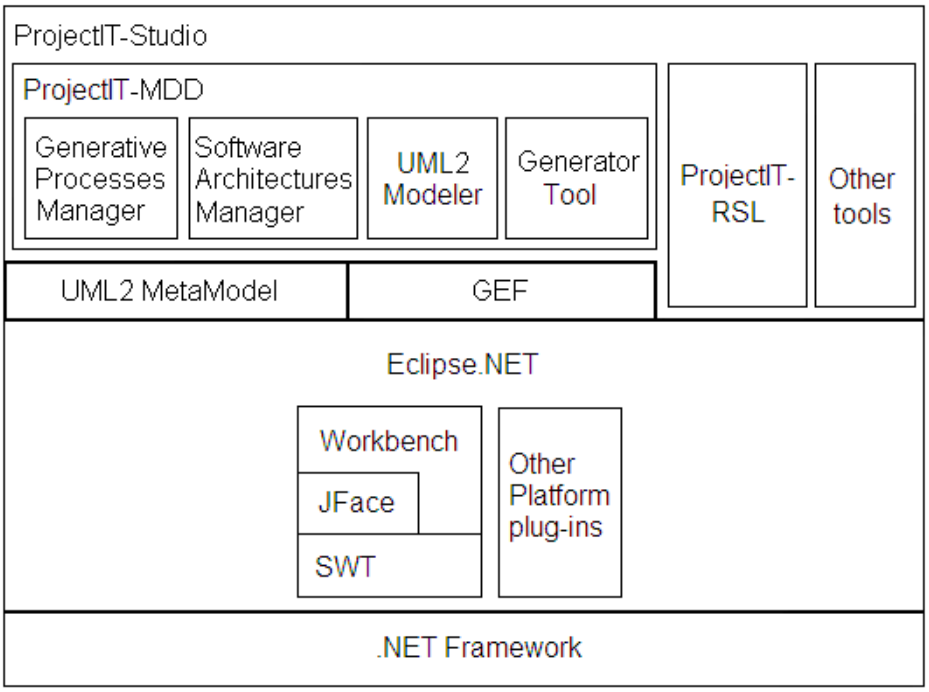

Figura 4.1 – Esquema do ProjectIT-*Studio* e os seus módulos, sobre a plataforma Eclipse.NET.

A cada ferramenta (ProjectIT-MDD, ProjectIT-RSL (*Requirements Specification Language*) [Silva2004], etc.) irá corresponder um ou mais módulos (poderá haver mais que um módulo por ferramenta por motivos de reaproveitamento de código; isto permite que

diferentes ferramentas possam utilizar o mesmo código – presente num módulo –, o que facilitará o desenvolvimento, depuração e manutenção do ProjectIT-*Studio*). Como exemplo, pretende-se a criação de um módulo que sirva como modelador UML2 (e portanto esse módulo irá importar um outro que implemente o meta-modelo do UML2) e que poderá comunicar com o módulo gerador de código; estes dois módulos irão posteriormente constituir o ProjectIT-MDD. A Figura 4.2 apresenta um diagrama de pacotes da ferramenta ProjectIT-Studio, ilustrando mais alguns módulos.

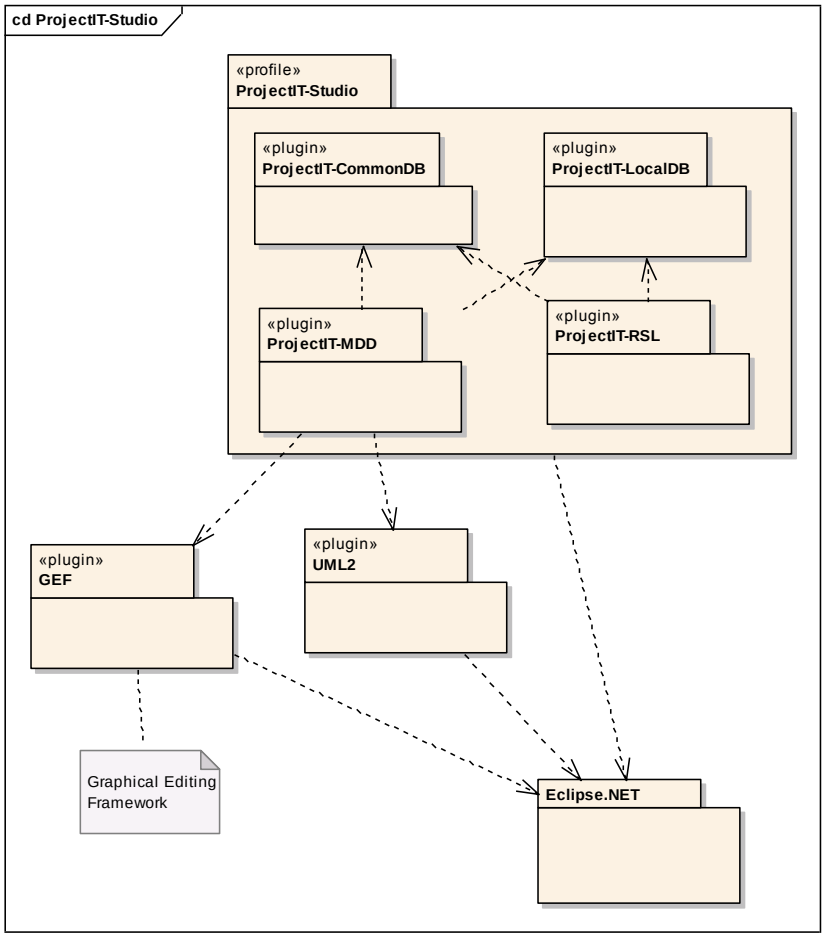

Figura 4.2 – Diagrama de pacotes do ProjectIT-*Studio*.

Note-se que os módulos ProjectIT-*CommonDB* e ProjectIT-*LocalDB* não correspondem a bases de dados, mas sim aos módulos que ficarão responsáveis pela sincronização com as bases de dados homónimas. O módulo GEF (*Graphical Editing Framework*) [EclipseGEF]

corresponde a um módulo que fornece o suporte para módulos que queiram incluir edição gráfica de modelos (como, por exemplo, um modelador UML).

O desenvolvimento da ferramenta ProjectIT-*Studio* sobre a plataforma Eclipse.NET terá várias tarefas, entre as quais: (1) a criação de um Perfil (designado "*ProjectIT.Studio*") que terá como função agrupar todos os módulos da ferramenta ProjectIT-*Studio* e personalizar a plataforma Eclipse.NET; (2) definir como é que os vários módulos do ProjectIT-*Studio* irão comunicar entre si, e declarar os pontos de extensão e as extensões para reflectir essa comunicação; (3) criar um módulo, baseado no módulo "*Core.Resources*", para fornecer os conceitos básicos de um "projecto ProjectIT"; e (4) desenvolvimento (e/ou adaptação) dos módulos propriamente ditos.

O Perfil criado irá especificar quais os módulos a serem carregados aquando da utilização da plataforma Eclipse.NET com a funcionalidade do ProjectIT-Studio, i.e., definir os módulos que serão carregados quando o utilizador inicia a plataforma escolhendo o Perfil "*ProjectIT.Studio*". Será também criado o módulo "*ProjectIT.Studio*", que será o módulo correspondente ao Perfil criado; este módulo não terá bibliotecas de código, apenas irá fornecer imagens e logótipos do ProjectIT, mensagens para a janela de "*About*", fornecer o nome da ferramenta ("ProjectIT-*Studio*") e outros recursos que sejam considerados necessários.

Será preciso definir como irão os vários módulos comunicar entre si, i.e., definir para cada módulo: (1) os módulos com que pode comunicar (para que módulos pode enviar informação e de que módulos a pode receber); e (2) o conteúdo da comunicação (que informação deve ser trocada durante a comunicação entre os dois módulos). Após essa planificação será possível declarar os pontos de extensão necessários para implementar essas comunicações. A comunicação entre dois módulos será, na maioria dos casos, feita recorrendo ao padrão "*Observer*" [Gamma1995] [Gamma2003], como ilustrado na Figura 4.3.

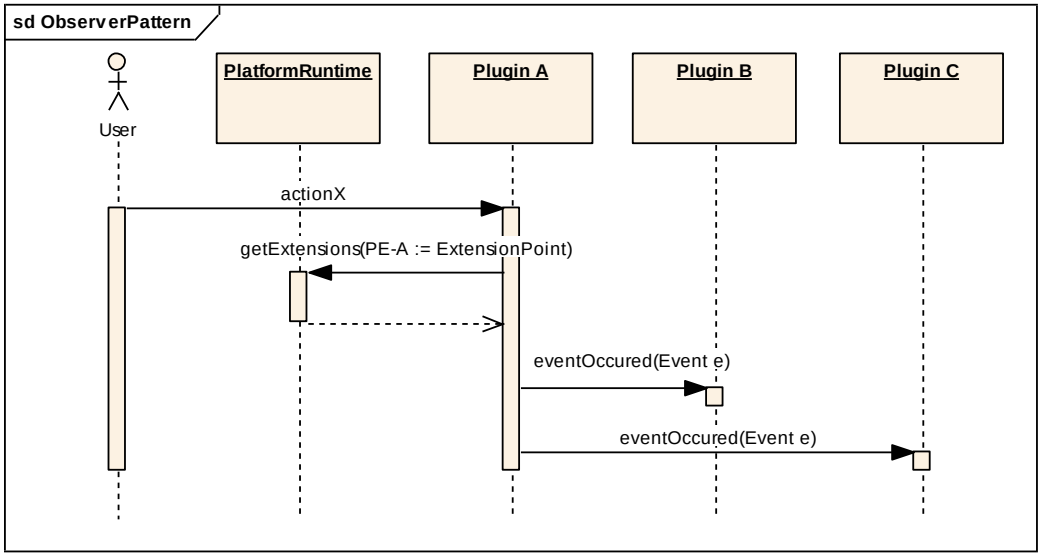

Figura 4.3 – Diagrama de sequências da comunicação entre módulos.

Seguindo o exemplo da Figura 4.3: suponha-se que o Módulo A declara um ponto de extensão, designado PE-A; suponha-se também que os Módulos B e C contribuem para esse ponto de extensão PE-A, cada um dos módulos com uma classe (que irá ter o papel de "*Observer*"). Quando o utilizador realizar a acção X ("*actionX*"), o Módulo A irá pedir à Plataforma (onde está localizado o Registo de Módulos) todas as extensões contribuídas para o ponto de extensão PE-A. Quando a Plataforma devolver ao Módulo A a lista de extensões contribuídas, o Módulo A irá obter, para cada extensão, o nome da classe que deve ser notificada da ocorrência da acção X e irá invocar o método "*eventOcurred*" dessa classe. Esse método, por sua vez, irá desencadear alguma acção (por exemplo, uma resposta ao utilizador) no módulo da classe a que pertence.

Será também necessário criar um módulo, baseado no módulo "*Core.Resources*" (que fornece os conceitos de "ficheiro", "directório" e "projecto" à plataforma Eclipse), que fornecerá o conceito de "projecto do ProjectIT" e "ficheiro" (de cada um dos componentes do ProjectIT-*Studio*: por exemplo, "ficheiro de modelo" para o ProjectIT-MDD, "ficheiro de especificação de requisitos" para o ProjectIT-RSL, etc.), bem como outros conceitos que se verifiquem necessários. Esta tarefa inclui também a definição de formatos dos ficheiros a utilizar (possivelmente com a definição de ficheiros XSD – *XML Schema Definition* – para especificar os conteúdos permitidos nos ficheiros-fonte do projecto, caso se opte pelo

armazenamento de dados em XML) e a definição de extensões de cada tipo de ficheiros (por exemplo, a extensão "*.mdl*" para ficheiros de modelo, a extensão "*.rsl*" para os ficheiros de especificação de requisitos ou a extensão "*.pit*" para os ficheiros com informação sobre o "projecto do ProjectIT").

Para além dos pontos de extensão declarados pelos módulos do ProjectIT-*Studio*, será preciso também definir alguns pontos de extensão da própria ferramenta ProjectIT-*Studio* (por exemplo, um ponto de extensão que indique os compiladores disponíveis e quais os tipos de ficheiro que cada compilador poderá receber para efectuar a sua compilação), de modo a que os vários módulos (ProjectIT-MDD, ProjectIT-RSL) possam ser adicionados ou removidos sem ter que alterar o próprio ProjectIT-*Studio*.

4.2 Evolução do *XIS-Tool* para o ProjectIT-*Studio*

Para uma melhor percepção do processo de adaptação da ferramenta *XIS-Tool*, é preciso primeiro explicar a maneira como esta foi originalmente desenvolvida.

A ferramenta *XIS-Tool* foi desenvolvida originalmente como uma aplicação *Web* em Java [Lemos2003], em conjunto com um modelo UML (no Perfil UML/XIS) da própria ferramenta *XIS-Tool*.

Mais tarde, esta foi convertida (com o auxílio da ferramenta JLCA) para uma ferramenta com interface gráfica *WinForms* [Queiroga2004]. Ainda no contexto desse trabalho, foram criados os *templates* para a criação de uma aplicação *WinForms*. Finalmente, foi fornecido ao *XIS-Tool* o modelo da ferramenta e os *templates* criados, para obter a nova versão do próprio *XIS-Tool*. Deste modo, a ferramenta *XIS-Tool* foi gerada por si própria. Para mais detalhes sobre o desenvolvimento da ferramenta até esta fase, aconselha-se a consulta de [Queiroga2004] e [Lemos2003].

No contexto deste trabalho, foi adaptada a ferramenta *XIS-Tool* para a nova plataforma. Foram criados os *templates* necessários para a criação de novos módulos a partir de modelos UML, segundo a Abordagem XIS, e foi também criado um novo módulo que desempenha as mesmas funções que a versão *WinForms* da ferramenta *XIS-Tool*.

De um modo geral, o desenvolvimento deste módulo apenas envolveu a criação dos novos *templates* relativos à interface gráfica (SWT). Todos os elementos da arquitectura de

"Módulo para Eclipse.NET" (excepto os elementos da interface gráfica) foram obtidos a partir dos elementos da arquitectura *WinForms* para a plataforma .NET, o que permitiu a reutilização de elementos dessa arquitectura. As arquitecturas *WinForms* e Eclipse.NET são descritas em maior detalhe no Anexo D.

Após a definição desses *templates*, a geração da ferramenta foi, mais uma vez, feita pela própria: apenas foi preciso gerar o módulo (recorrendo ao modelo UML definido em [Lemos2003] e aos *templates* definidos para a arquitectura Eclipse.NET). Foi também preciso reaproveitar a lógica presente na ferramenta *XIS-Tool* (relativa às funcionalidades de geração e importação), pois essa não pôde ser gerada a partir do modelo UML.

A Figura 4.4 e a Figura 4.5 ilustram a situação (na ferramenta *XIS-Tool* e na ferramenta ProjectIT-*Studio*) em que o arquitecto está a ver o ecrã de edição dos processos de geração, onde pode começar um dos processos de geração, por exemplo.

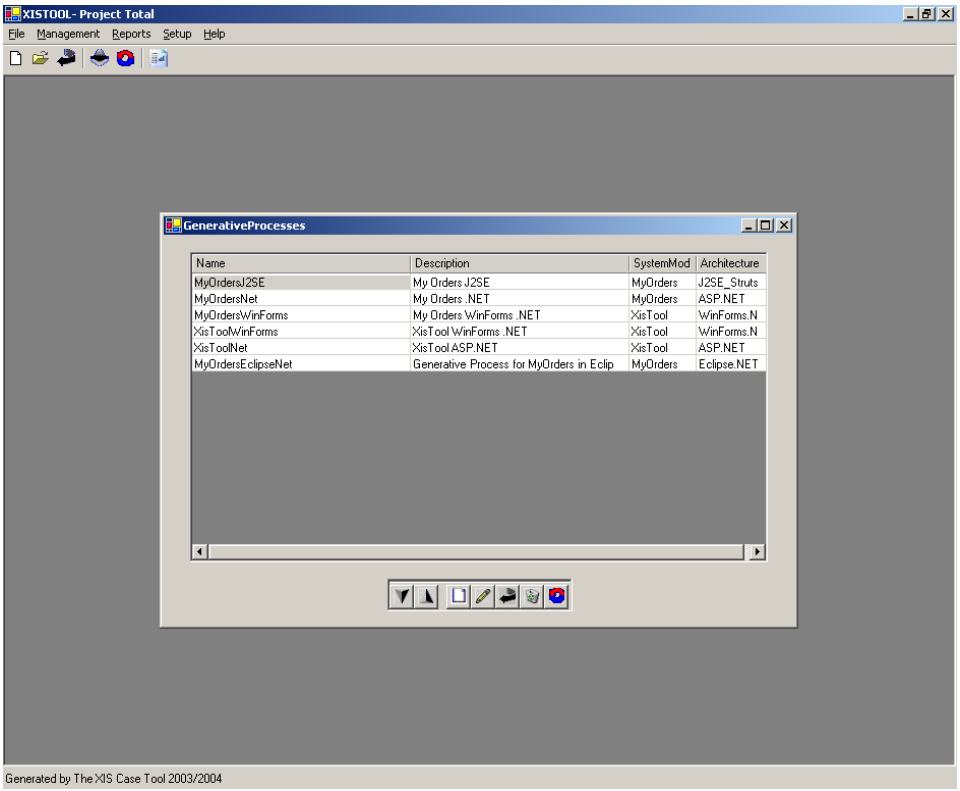

Figura 4.4 – Ferramenta *XIS-Tool* com o editor de todos os processos de geração.

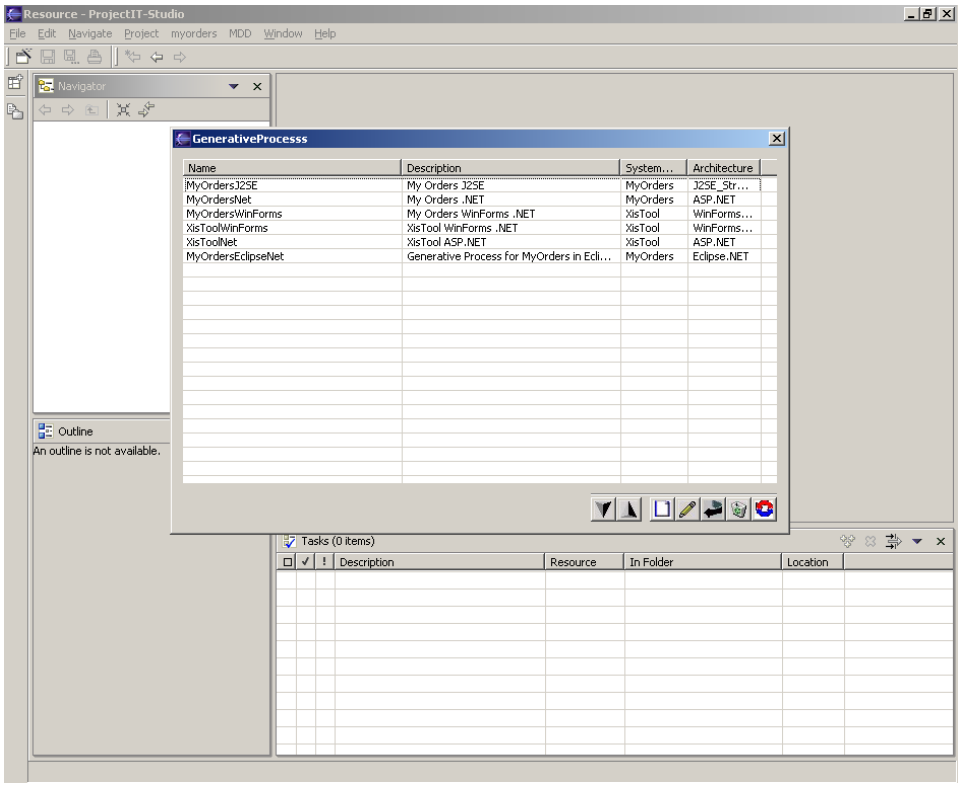

Figura 4.5 – Editor de todos os processos de geração, mas na ferramenta ProjectIT-*Studio*.

A Figura 4.6 e a Figura 4.7 ilustram a situação (na ferramenta *XIS-Tool* e na ferramenta ProjectIT-*Studio*, respectivamente) em que o arquitecto está a ver um relatório dos processos de geração disponíveis no projecto aberto; caso o arquitecto queira ver um relatório mais detalhado sobre um certo processo de geração, terá apenas que carregar no ícone .

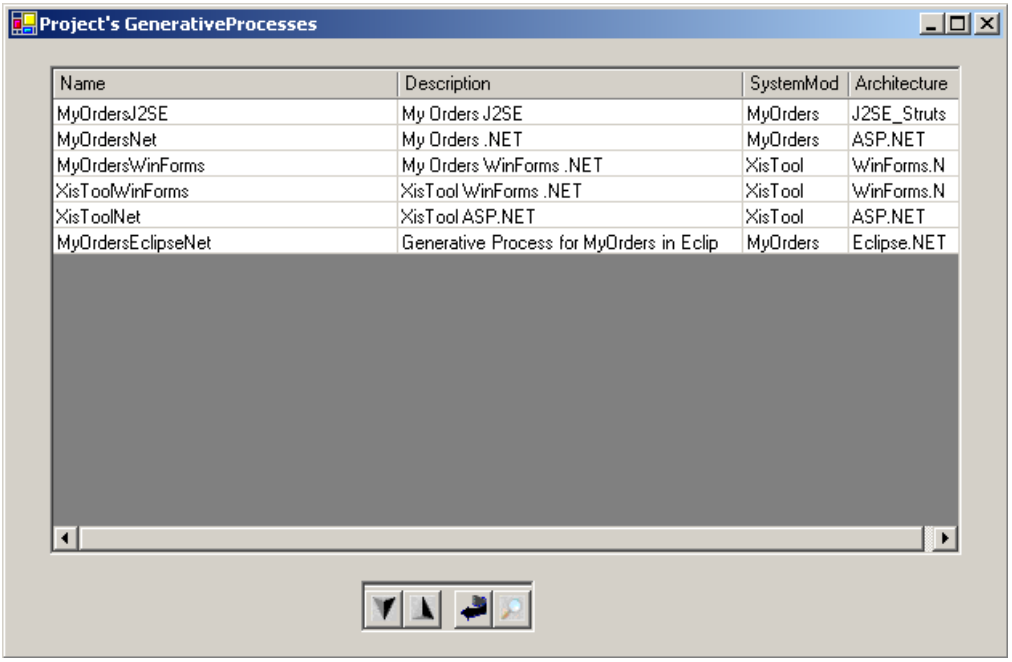

Figura 4.6 – Ferramenta *XIS-Tool* a mostrar alguns processos de geração.

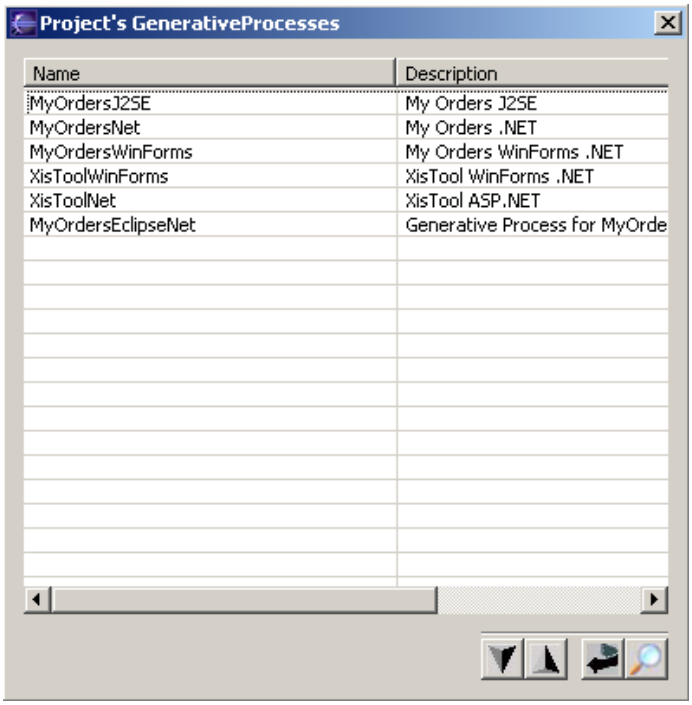

Figura 4.7 – Os processos de geração da Figura 4.6, mas na ferramenta ProjectIT-*Studio*.

5 Conclusões

Neste documento foi descrito o trabalho desenvolvido no âmbito deste TFC, com ênfase nos conceitos e técnicas empregues e o modo como estas foram usadas de modo a resolver os problemas postos.

Antes do início deste trabalho (em Setembro de 2004), a implementação do ProjectIT-*Studio* consistia num conjunto de ferramentas, cuja comunicação era feita apenas com base em ficheiros e com a intervenção directa do utilizador da ferramenta (i.e., o utilizador tinha que guardar em ficheiro os resultados produzidos por uma ferramenta A, iniciar outra ferramenta B e abrir nessa nova ferramenta B os dados produzidos pela ferramenta A), e cujo nível de integração (e automatização de tarefas) era claramente insuficiente. A Figura 5.1 apresenta o modo como o ProjectIT-*Studio* estava então a ser desenvolvido (para simplicidade da figura, só são ilustrados alguns dos componentes do ProjectIT-*Studio*).

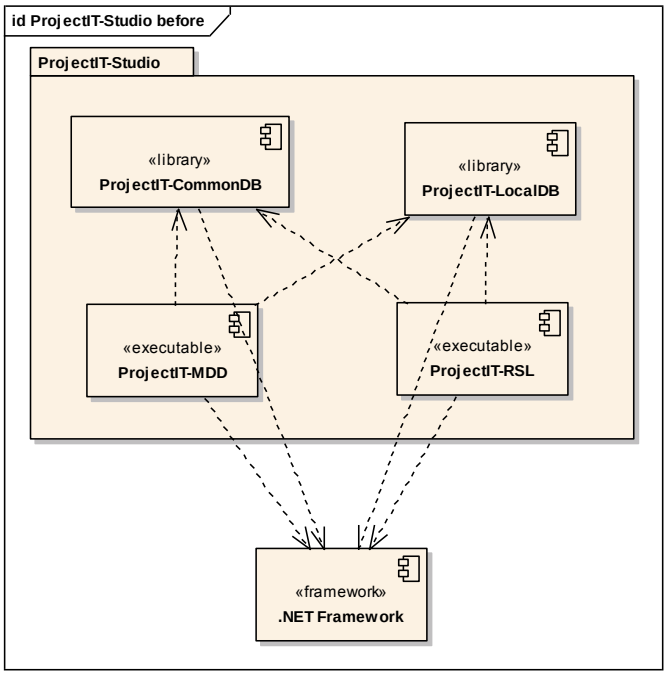

Figura 5.1 – Desenvolvimento do ProjectIT-*Studio*, antes deste trabalho.

Este trabalho, enquadrado no projecto de investigação ProjectIT, propõe uma solução para o problema da integração entre as diferentes ferramentas desenvolvidas no contexto do ProjectIT-*Studio*. A solução proposta consiste em: (1) criação da plataforma Eclipse.NET (uma conversão da plataforma Eclipse para .NET); (2) customização da plataforma para a ferramenta ProjectIT-Studio; e (3) posterior adaptação das ferramentas do ProjectIT-Studio para Módulos da nova plataforma.

Embora a solução proposta pareça demasiado ambiciosa e complexa a curto prazo, a longo prazo previne problemas que iriam inevitavelmente ocorrer quando se tentasse integrar as várias ferramentas. É também de realçar que esta solução promove a modularização e a reutilização de código, colocando código usado frequentemente por vários módulos num outro módulo e estabelecendo relações de importação entre esses módulos. Com esta solução, a comunicação entre as várias ferramentas do ProjectIT-*Studio* poderá ser baseada no mecanismo de Pontos de Extensão oferecido pela plataforma Eclipse.NET, o que irá contribuir para a automatização de tarefas e para um melhor nível de integração das ferramentas do ProjectIT-*Studio*, sem quebra da modularidade entre essas mesmas ferramentas.

Durante este TFC foi também desenvolvida a plataforma, que servirá como ferramenta para integração no âmbito do .NET. O Eclipse.NET é uma plataforma inédita na comunidade .NET, e poderá ter grande potencial para evolução, tanto dentro como fora do contexto do ProjectIT.

Foram também convertidos alguns módulos que serão muito úteis no futuro. Entre estes encontra-se o módulo GEF (*Graphical Editing Framework*) [EclipseGEF], que consiste num módulo para criar editores gráficos com base no modelo MVC (*Model-View-Controller*).

Após esta conversão, a plataforma Eclipse.NET foi customizada para a ferramenta ProjectIT-*Studio*. Foram também integradas na plataforma, de forma preliminar, algumas funcionalidades, em particular as existentes no âmbito da ferramenta *XIS-Tool*, agora designada ProjectIT-MDD. Foram criados os *templates* necessários para criar um módulo para a plataforma Eclipse.NET a partir de um modelo, à imagem do que aconteceu com o desenvolvimento do próprio *XIS-Tool* na sua versão anterior.

A Figura 5.2 apresenta o planeamento da maneira como o ProjectIT-*Studio* irá ser desenvolvido após este trabalho.

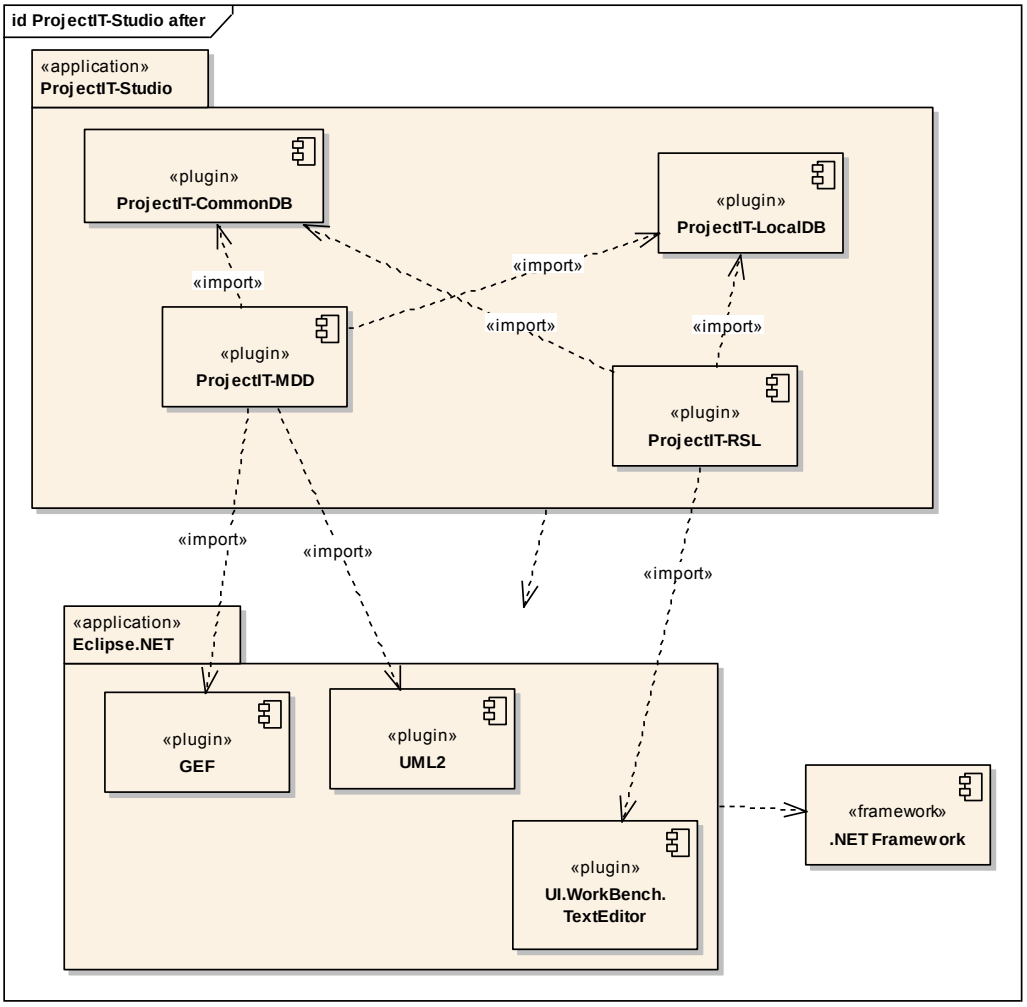

Figura 5.2 – Desenvolvimento do ProjectIT-*Studio* depois deste trabalho.

A Figura 5.2 não ilustra todos os módulos do ProjectIT-*Studio*, para simplicidade da figura. Note-se também que, embora os módulos UML2 e GEF não estejam incluídos na Plataforma Eclipse (i.e., não são desenvolvidos no contexto da Plataforma), a figura apresenta esses módulos dentro do pacote Eclipse.NET, com o objectivo de melhor exemplificar a dependência do ProjectIT-*Studio* da plataforma Eclipse.NET.

A visão para o futuro, e para a qual este trabalho espera servir como catalizador, é que a plataforma Eclipse.NET seja aceite pela comunidade .NET e que esta aceite a tarefa e a responsabilidade de assegurar o esforço de manutenção, desenvolvimento e evolução da plataforma. O ProjectIT-*Studio* também irá acompanhar essa evolução, melhorando as suas ferramentas e adoptando as inovações que venham a ser incluídas na plataforma. Espera-se também que outras aplicações sigam o exemplo do ProjectIT-*Studio* e que sejam desenvolvidas para a plataforma Eclipse.NET, de forma a torná-la um *standard* na comunidade .NET e um veículo para a reutilização de funcionalidades e para a promoção de boas práticas, um factor para o qual a indústria das Tecnologias da Informação só recentemente começou a dar a devida atenção.

A nível pessoal, esta experiência foi bastante enriquecedora e algo fora do comum. Foi muito interessante verificar como foi desenvolvida e funciona realmente uma ferramenta como o Eclipse, que muita gente usa no seu dia-a-dia mas que muito poucos conhecem os seus conceitos e arquitectura. Foi também interessante verificar que as boas práticas, leccionadas ao longo do curso, têm realmente razão de ser!

5.1 Críticas ao trabalho efectuado

Como já foi referido, a plataforma desenvolvida tem potencial. Contudo, a plataforma e os módulos desenvolvidos ainda se encontram num estado muito básico e há muito trabalho a fazer. É preciso corrigir ainda muitas falhas, resultado de diferenças (algumas subtis, outras mais radicais) entre a API da plataforma Java e a API da plataforma .NET. Durante o decorrer deste trabalho, foram corrigidas algumas dessas falhas. Contudo, a complexidade e a quantidade do código envolvido tornaram esta tarefa muito morosa e muito difícil de realizar nos cerca de 8 meses que durou este TFC.

A própria conversão das funcionalidades da ferramenta *XIS-Tool*, da sua versão *WinForms*, para a versão Módulo Eclipse.NET, precisa de ser melhorada. O trabalho desenvolvido no contexto desta ferramenta consistiu apenas na criação de novos *templates* para suportar a arquitectura da ferramenta Eclipse.NET; contudo, não foi introduzida nenhuma melhoria, o que deixa esta parte do trabalho em aberto para o futuro.

5.2 Trabalho futuro (Eclipse.NET e ProjectIT-*Studio*)

A plataforma Eclipse.NET ainda tem um longo caminho pela frente. Não só é preciso corrigir as falhas existentes, como é preciso fazê-la evoluir, adicionando novas ideias e tornando-a uma plataforma mais estável, rápida e fácil de utilizar.

Também será necessário actualizar os módulos existentes, à medida que a plataforma evolua, devido a mudanças na API, etc.

Será preciso ainda converter alguns módulos (não essenciais para a plataforma, mas que são essenciais para o desenvolvimento baseado nesta, como é o caso do módulo de suporte à depuração – "*Core.Debug*") que não foram convertidos no contexto deste trabalho.

Finalmente, será preciso começar a migrar as ferramentas existentes no contexto do ProjectIT-*Studio* para a nova plataforma. Prevê-se que uma parte significativa do código existente (parte essa relativa à lógica de negócio) possa ser directamente reutilizada, sendo necessário alterar apenas o código relativo à interface com o utilizador (passar o código de *WinForms* para SWT [EclipseSWT]). Contudo, será também preciso definir os Pontos de Extensão a utilizar no contexto do ProjectIT-Studio. Será preciso também definir a semântica desses mesmos Pontos de Extensão (i.e., definir o que cada um espera receber – interfaces e classes a utilizar –, como irá ser usado cada um desses Pontos de Extensão, que módulos devem fornecer Pontos de Extensão e quais os módulos que devem contribuir Extensões para esses Pontos de Extensão, etc.).

5.3 Trabalho a desenvolver no âmbito do Mestrado

No âmbito do Mestrado a realizar no próximo ano são delineados os seguintes objectivos de trabalho:

- **Criar um modelador UML com capacidades de extensão:** pretende-se criar um modelador UML que seja facilmente extensível (por estereótipos, perfis UML, etc.). Planeia-se adaptar posteriormente este modelador ao perfil ProjectIT/UML, de modo a permitir especificar modelos que possam, por exemplo, ser fornecidos ao módulo ProjectIT-MDD;

- **Alterar a abordagem e ferramenta ProjectIT-*Studio* para facilitar a adição de *templates* e arquitecturas:** actualmente, a abordagem ProjectIT-MDD oferece um mecanismo que permite ao utilizador especificar *templates* e arquitecturas. Contudo, a interface deste mecanismo é de utilização morosa e propensa a erros por parte do arquitecto de *software*. Pretende-se a melhoria dessa interface e, eventualmente, uma alteração da própria abordagem para permitir uma especificação mais fácil e eficiente desses *templates* e arquitecturas;
- **Melhorar a usabilidade da ferramenta/módulo ProjectIT-MDD:** a interface da ferramenta ProjectIT-MDD continua a não ser muito usável. Obviamente, este factor é muito importante para a eficácia e aceitação desta ferramenta por parte dos utilizadores. Pretende-se melhorar a usabilidade da ferramenta, recorrendo a várias boas práticas no campo da Interacção Pessoa-Máquina;
- **Estudar mecanismos de armazenamento dos modelos criados pelo ProjectIT-*Studio*:** pretende-se estudar mecanismos para armazenar modelos criados pelas ferramentas do ProjectIT-*Studio* (nomeadamente, pelo modelador ProjectIT/UML), por exemplo, em ficheiros XML ou em ficheiros em formato *Microsoft Access*;
- **Estudar mecanismos de sincronização entre as bases de dados ProjectIT-*LocalDB* e ProjectIT-*CommonDB*:** pretende-se também estudar mecanismos para permitir sincronização correcta e eficiente entre as bases de dados ProjectIT-*LocalDB* e ProjectIT-*CommonDB* (ver Figura 1.2). Este mecanismo permitirá à equipa de desenvolvimento trabalhar sobre uma versão local dos modelos (o ProjectIT-*LocalDB*) e, posteriormente, sincronizar o trabalho realizado com o trabalho realizado por todas as outras equipas, num repositório centralizado (o ProjectIT-*CommonDB*).

6 Referências

6.1 Referências bibliográficas

[Archer2001] T. Archer, *Inside C#*, Microsoft Press, 2001.

[Bass2003] L. Bass et al. *Software Architecture in Practice, Second Edition*, Addison-Wesley, 2003.

[Clements2002] P. Clements et al. *Documenting Software Architectures: Views and Beyond*, Addison-Wesley, 2002.

[Faria2003] J. A. Faria. "*Business Change Modeller. Relatório de Trabalho Final de Curso*". Instituto Superior Técnico, Julho 2003.

[Fowler2003] M. Fowler et al, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003.

[Gamma1995] E. Gamma et al, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Gamma2003] E. Gamma e K. Beck, *Contributing to Eclipse: Principles, Patterns, and Plug-Ins*, Addison-Wesley, 2003.

[Lemos2003] G. Lemos e T. Matias. "*Relatório do TFC Projecto XIS – Abordagem e Ferramenta de Desenvolvimento (Case Tool e UML)*". Instituto Superior Técnico, Julho 2003.

[McGraw1999] G. McGraw e E. Felten, *Securing Java*. John Wiley & Sons Inc., 1999. (também disponível em http://www.securingjava.com/chapter-two/chapter-two-7.html)

[Queiroga2004] R. B. Queiroga. "*The XIS CASE Tool. Relatório de Trabalho Final de Curso*". Instituto Superior Técnico, Outubro 2004.

[Saraiva2005] J. S. Saraiva e A. R. Silva. Eclipse.NET: An Integration Platform for ProjectIT-Studio. Em *Proceedings of the First International Conference of Innovative Views of .NET Technologies*. Instituto Superior de Engenharia do Porto, Junho 2005.

[Silva2001] A. R. Silva e C. Videira, *UML, Metodologias e Ferramentas CASE*. Centro Atlântico, Portugal, 2001.

[Silva2003a] A. R. Silva. The XIS Approach and Principles. Em *Proceedings of the 29th EUROMICRO Conference*. IEEE Computer Society, Setembro 2003.

[Silva2003b] A. R. Silva et al. The XIS Generative Programming Techniques. Em *Proceedings of the 27th COMPSAC Conference*. IEEE Computer Society, Novembro 2003.

[Silva2004] A. R. Silva. "*O Programa de Investigação ProjectIT*". INESC-ID & Instituto Superior Técnico, Outubro 2004. (também disponível em http://berlin.inesc-id.pt/alb/uploads/1/193/pit-white-paper-v1.0.pdf)

[Silva2005a] A. R. Silva. "*Programa de I&D ProjectIT*". INESC-ID & Instituto Superior Técnico, Abril 2005. (também disponível em http://berlin.inesc-id.pt/alb/uploads/1/438/pit-status-2005-v0.1.pdf)

[Silva2005b] A. R. Silva e C. Videira, *UML, Metodologias e Ferramentas CASE – 2ª Edição – Volume 1*. Centro Atlântico, Portugal, 2005.

6.2 Referências na Internet

[.NET] "*Microsoft .NET Homepage*"
http://www.microsoft.com/net

"*MSDN .NET Framework Developer Center: Technology Overview*"
http://msdn.microsoft.com/netframework/technologyinfo/overview/

"*Overview of the .NET Framework*"
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpovrintroductiontonetframeworksdk.asp

[ArgoUML] "*ArgoUML Project home*"
http://argouml.tigris.org

[ASP.NET] "*ASP.NET Web: The Official Microsoft ASP.NET Site : Home Page*"
http://www.asp.net/

[Azureus] "*Azureus: Java BitTorrent Client*"
http://azureus.sourceforge.net/

[CEO] "CEO – Centro de Engenharia Organizacional"
http://ceo.inesc.pt/index.shtml

[Eclipse] "*Eclipse.org Main page*"
http://www.eclipse.org

"*Eclipse Platform Technical Overview"* (Fevereiro 2003)
http://www.eclipse.org/whitepapers/eclipse-overview.pdf

Apresentação (em diapositivos) do projecto Eclipse (2003)
http://www.eclipse.org/eclipse/presentation/eclipse-slides.pdf

[EclipseEMF] "*Eclipse Tools – EMF Home*"
http://www.eclipse.org/emf/

[EclipseGEF] "*The Eclipse Graphical Editing Framework Project*"
http://www.eclipse.org/gef

[EclipseSWT] "*SWT: The Standard Widget Toolkit*"
http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/platform-swt-home/main.html

[EclipseUML2] "*The Eclipse UML2 Project*"
http://www.eclipse.org/uml2

[Gunnerson2002] E. Gunnerson, "*AppDomains and Dynamic Loading*". Maio 2002.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncscol/html/csharp05162002.asp

[IKVM.NET] "*IKVM.NET Home Page*"
http://www.ikvm.net

"*IKVM.NET Weblog*"
http://weblog.ikvm.net/

[INESC-ID] INESC-ID, Grupo de Sistemas de Informação.
Web site do *ProjectIT*: http://berlin.inesc-id.pt/alb/ProjectIT@81.aspx

[Java] "*Java Technology*"
http://java.sun.com

[Jeffries2001] R. Jeffries, "*What is Extreme Programming?*". Agosto 2001.
http://www.xprogramming.com/xpmag/whatisxp.htm

[JLCA] "*Microsoft Java Language Conversion Assistant 3.0 beta*"
http://msdn.microsoft.com/vstudio/downloads/tools/jlca/30beta/default.aspx

[Log4NET] "*log4NET – log4NET Project*"
http://logging.apache.org/log4net/

[MDA] "*OMG Model Driven Architecture*"
http://www.omg.org/mda

[Mono] "*Mono*"
http://www.mono-project.com

[MonoUML] "*MonoUML Case Tool*"
http://www.monouml.org

[MSVisual Studio] "*Visual Studio Home*"
http://msdn.microsoft.com/vstudio/

[NetBeans] "*Welcome to NetBeans*"
http://www.netbeans.org/

[OMG] "*Object Management Group*"
http://www.omg.org/

[OSGi] "*The OSGi Service Platform*"
http://www.osgi.org

[Renaud2003] J. Renaud, "*Undocumented Fusion*". Julho 2003.
http://www.codeproject.com/dotnet/undocumentedfusion.asp

[SeminárioDesignPatterns] "Seminário: *Design Patterns*" http://eden.dei.uc.pt/~barros/DesignPatterns2005.htm

J. Saraiva. "*Design Patterns na plataforma Java: Uma experiência no processo de migração para .NET*". INESC-ID & Instituto Superior Técnico, Março 2005. http://eden.dei.uc.pt/~barros/JoaoSaraiva.pdf

[Shah2003] R. Shah, "*Working the Eclipse Platform*". Março 2003. http://www-106.ibm.com/developerworks/opensource/library/os-eclipse

[Thorn] "*SourceForge.net: Project Info - Thorn*" http://sourceforge.net/projects/thorn/

[UML] "*Unified Modelling Language*" http://www.uml.org

[W3C] "*World Wide Web Consortium*" http://www.w3.org/

[XMI] "*XML Metadata Interchange*" http://www.omg.org/technology/documents/formal/xmi.htm

[XML] "*Extensible Markup Language (XML)*" http://www.w3.org/XML/